Categorical Foundations for CuTe Layouts

Colfax Research

September 2025

Abstract

NVIDIA's CUTLASS library provides a robust and expressive set of methods for describing and manipulating multi-dimensional tensor data on the GPU. These methods are conceptually grounded in the abstract notion of a CuTe layout and a rich algebra of such layouts, including operations such as composition, logical product, and logical division. In this paper, we present a categorical framework for understanding this layout algebra by focusing on a naturally occurring class of *tractable layouts*. To this end, we define two categories **Tuple** and **Nest** whose morphisms give rise to layouts. We define a suite of operations on morphisms in these categories and prove their compatibility with the corresponding layout operations. Moreover, we give a complete characterization of the layouts which arise from our construction. Finally, we provide a Python implementation of our categorical constructions, along with tests that demonstrate alignment with CUTLASS behavior. This implementation can be found at our git repository https://github.com/ColfaxResearch/layout-categories.

Contents

Chapter 1

Introduction

In modern computing, particularly in GPU programming, performance depends critically on how multi-dimensional data is stored and accessed in memory. While most data that we care about—such as images, videos, and tensors in machine learning—are inherently multi-dimensional, a computer's memory is fundamentally one-dimensional. This means that when we want to load, store, or otherwise manipulate data, we need to map its multi-dimensional logical coordinates to one-dimensional physical coordinates. This mapping, known as a **layout**, is essential for reading from and writing to memory correctly and efficiently. Moreover, with respect to the GPU's SIMT execution model, layouts are used to describe and manipulate partitionings of threads over data. This is important to ensure optimized memory access patterns and correct invocation of specialized hardware instructions such as those used to target tensor cores.

As a motivating example, suppose we want to store the 4×8 matrix

$$A = \begin{bmatrix} 12.47 & 87.21 & 34.08 & 56.93 & 45.65 & 9.17 & 73.02 & 21.39 \\ 64.88 & 30.41 & 1.72 & 88.04 & 92.55 & 17.06 & 50.91 & 68.77 \\ 3.33 & 77.19 & 61.58 & 29.46 & 15.82 & 80.75 & 44.62 & 39.28 \\ 91.40 & 26.12 & 6.97 & 53.03 & 58.66 & 33.79 & 11.20 & 70.55 \end{bmatrix}$$

in memory. In order to do so, we need to specify a memory address for each entry of A. We do this by choosing some address for the $(0,0)$th entry of A, and specifying an **offset** for each other entry of A. One common choice is the **row-major** layout

$$L^{\mathsf{row}} = (4,8) : (8,1) =$$

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

The notation $L^{\mathsf{row}} = (4,8) : (8,1)$ indicates that the offset of the (i,j)th entry of our matrix is

$$(i,j) \cdot (8,1) = 8i + j.$$

Another common choice is the **column-major** layout

$$L^{\mathsf{col}} = (4, 8) : (1, 4) =$$

0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

Again, the notation $L^{\mathsf{col}} = (4, 8) : (1, 4)$ indicates that the offset of the (i, j)th entry of our matrix is

$$(i, j) \cdot (1, 4) = i + 4j.$$

These layouts are extremely useful, but do not suffice for all purposes. For example, in high-performance computing, one often computes matrix products AB by

1. dividing the operand matrices A and B into tiles,

2. computing matrix products of the various tiles, and

3. combining these partial results to obtain the full result AB.

For instance, we could divide our 4×8 matrix A into 2×2 tiles, as depicted below.

$$A = \begin{bmatrix} \begin{bmatrix} 12.47 & 87.21 \\ 64.88 & 30.41 \end{bmatrix} & \begin{bmatrix} 34.08 & 56.93 \\ 1.72 & 88.04 \end{bmatrix} & \begin{bmatrix} 45.65 & 9.17 \\ 92.55 & 17.06 \end{bmatrix} & \begin{bmatrix} 73.02 & 21.39 \\ 50.91 & 68.77 \end{bmatrix} \\ \begin{bmatrix} 3.33 & 77.19 \\ 91.40 & 26.12 \end{bmatrix} & \begin{bmatrix} 61.58 & 29.46 \\ 6.97 & 53.03 \end{bmatrix} & \begin{bmatrix} 15.82 & 80.75 \\ 58.66 & 33.79 \end{bmatrix} & \begin{bmatrix} 44.62 & 39.28 \\ 11.20 & 70.55 \end{bmatrix} \end{bmatrix}$$

Suppose now that we wanted to slice out individual tiles of A, which we assume is laid out in column-major format in memory. To do this, one could manually compute offsets as follows: for the (i, j)th tile, the offset to index into the top-left entry of the tile is given by $2i + 8j$. On the other hand, to better organize this computation, we could use the **interleaved** layout of tiles

$$L^{\mathsf{tiled}} =$$

0	2	8	10	16	18	24	26
1	3	9	11	17	19	25	27
4	6	12	14	20	22	28	30
5	7	13	15	21	23	29	31

where the columns are given by tiles of A and the rows are given by coordinates within the tile shape. Here, we use **colexicographic ordering** to linearly enumerate tiles and coordinates within tiles, hence the top-level shape $(4, 8)$ of the layout L^{tiled}.

However, note that the interleaving pattern shown for L^{tiled} means that it can't be expressed as a layout $(4, 8) : (a, b)$ for any strides a, b. Instead, we can factor the modes of the shape $(4, 8)$ and define

$$L^{\text{tiled}} = ((2, 2), (2, 4)) : ((1, 4), (2, 8)).$$

The prior offset calculation $2i + 8j$ then appears through evaluating L^{tiled} on the coordinate $(0, (i, j))$, and the tile layout itself is given by the first mode. Thus, after endowing A with the layout L^{tiled} to form A^{tiled}, we can obtain the (i, j)th tile of A as the slice

$$A_{i,j} = A^{\text{tiled}}(\, - \,, (i, j)).$$

A key idea developed in CUTLASS is that useful but more complex auxiliary layouts such as L^{tiled} may be systematically deduced from simpler layouts via certain fundamental operations. In the case of L^{tiled}, the operation in question is called **logical division**. If we write

$$T = (2, 2) : (1, 4) = \begin{array}{|c|c|} \hline 0 & 4 \\ \hline 1 & 5 \\ \hline \end{array}$$

for the tile layout, then L^{tiled} is the logical division

$$L^{\text{tiled}} = L^{\text{col}} \oslash T$$

as depicted below.

In addition to logical division, other fundamental layout operations include **logical products**, **complements**, and most importantly, **composition**. These layout operations are the backbone of CUTLASS, and a deep understanding of their behavior is helpful for writing correct and highly
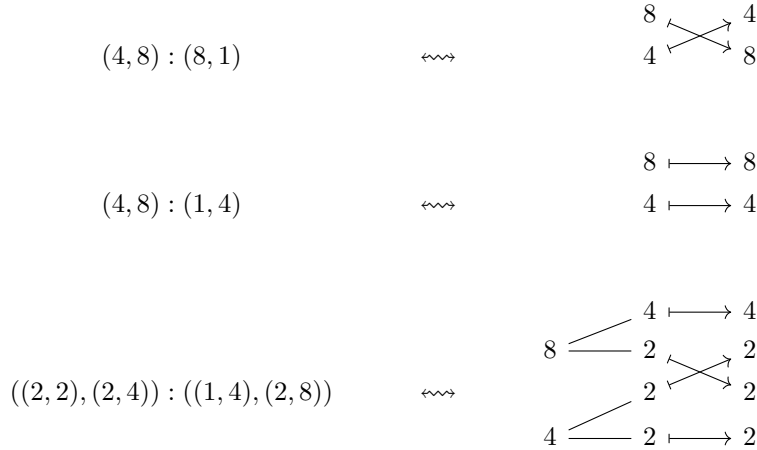
performant code. However, the definitions and constructions of these operations are fairly subtle. For example, the composition $B \circ A$ of layouts A and B is well-defined only if A and B satisfy certain divisibility constraints, which CUTLASS checks under the hood. In particular, it is not always obvious when two layouts are composable, or how to interpret their composition.

1.1 Summary of main results

The main idea of this work is that we can develop an intuitive and powerful mathematical framework for working with layouts by restricting our attention to **tractable layouts**, whose entries satisfy a simple divisibility condition (see Definition 2.3.10.1). Tractable layouts include almost all layouts one encounters in practice, such as

- *row-major* and *column-major* layouts, which are ubiquitous,

- *compact* layouts, which store data in consecutive memory addresses,

- *projections*, which broadcast multiple copies of data, and

- *dilations*, which enable padded loads and stores.

If L is a tractable layout, then we can represent L with a **diagram**. For example, the layouts L^{row}, L^{col}, and L^{tiled} are represented by the following diagrams.

$$(4,8):(8,1) \qquad \longleftrightarrow \qquad \begin{array}{ccc} 8 & & 4 \\ & \times & \\ 4 & & 8 \end{array}$$

$$(4,8):(1,4) \qquad \longleftrightarrow \qquad \begin{array}{ccc} 8 & \longmapsto & 8 \\ 4 & \longmapsto & 4 \end{array}$$

$$((2,2),(2,4)):((1,4),(2,8)) \qquad \longleftrightarrow \qquad \begin{array}{c} 4 \longmapsto 4 \\ 8 \underline{\quad} 2 \quad 2 \\ 2 \times 2 \\ 4 \underline{\quad} 2 \longmapsto 2 \end{array}$$

These diagrams may be interpreted as morphisms in a **category**. This allows us to leverage the power of **category theory** to describe layouts and their operations.[1]

More precisely, we define a category **Nest** whose objects are nested tuples of positive integers, and whose morphisms $f : S \to T$ correspond to diagrams such as those above (see Definition 3.1.1.13 and Definition 3.2.1.1 for details). If L is a **non-degenerate** tractable layout (see Definition 2.3.1.24), then there is an essentially unique **Nest**-morphism f which encodes L, as illustrated by the following correspondence theorem.

Theorem A. *(see 3.2.2.15) There is a one-to-one correspondence*

$$\left\{ \begin{array}{c} \textit{Non-degenerate} \\ \textit{tractable layouts} \end{array} \right\} \longleftrightarrow \left\{ \begin{array}{c} \textit{Non-degenerate} \\ \textbf{Nest}\textit{-morphisms} \\ \textit{of standard form} \end{array} \right\}$$

---

[1]We provide a primer on category theory in Appendix A for those unfamiliar with the subject.

Layout operations such as composition, logical division, and logical products may be interpreted naturally in the category **Nest**. If

$$S \xrightarrow{\ f\ } T \xrightarrow{\ g\ } U$$

are **Nest**-morphisms, then we may form the composite

$$S \xrightarrow{\ g \circ f\ } U$$

by pasting the associated diagrams together. For example,
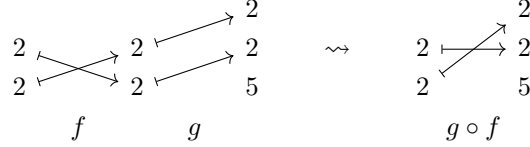
We prove that composition in **Nest** is compatible with layout composition.

Theorem B. *(see 3.2.6.21) If f and g are non-degenerate composable **Nest**-morphisms, then*

$$L_{g \circ f} = L_g \circ L_f.$$

We can coalesce a **Nest**-morphism f by collapsing adjacent arrows. For example,
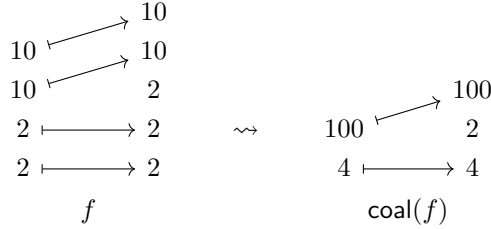
We prove that this operation is compatible with layout coalesce.

Theorem C. *(see 3.2.6.13) If f is a **Nest**-morphism, then*

$$L_{\mathsf{coal}(f)} = \mathsf{coal}(L_f).$$

The complement of a **Nest**-morphism f is the inclusion of the entries not hit by f. For example,
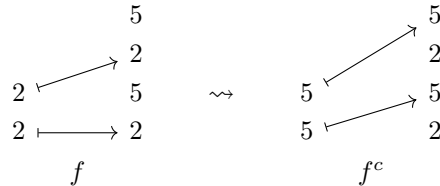
We prove that complements in **Nest** are compatible with layout complements.

Theorem D. *(see 3.2.6.20) If $f : S \to T$ is an injective **Nest**-morphism and $N = \mathsf{size}(T)$, then*

$$\mathsf{coal}(L_{f^c}) = \mathsf{comp}(L_f, N).$$

7

We define divisibility of **Nest**-morphisms, and a logical division operation

$$f, g \mapsto f \oslash g$$

when g divides f. For example,

$$
\begin{array}{ccc}
\begin{array}{l}
8 \longmapsto 8 \\
\quad 4 \longmapsto 4 \\
4 \longmapsto 8 \longmapsto 8 \\
4 \longmapsto 4 \longmapsto 4 \\
\quad g \qquad f
\end{array}
&
\rightsquigarrow
&
\begin{array}{l}
\quad\; 8 \longmapsto 8 \\
64 \longmapsto 8 \longmapsto 4 \\
\quad\; 4 \longmapsto 8 \\
16 \longmapsto 4 \longmapsto 4 \\
\qquad f \oslash g
\end{array}
\end{array}
$$

We prove that logical division in **Nest** is compatible with logical division of layouts.

Theorem E. *(see 3.2.6.26) If f and g are non-degenerate **Nest**-morphisms and g divides f, then*

$$\operatorname{coal}(L_{f \oslash g}) = \operatorname{coal}(L_f \oslash L_g).$$

We define product admissibility of **Nest**-morphisms, and a logical product operation

$$f, g \mapsto f \otimes g$$

when f and g are product admissible. For example,

$$
\begin{array}{ccc}
\begin{array}{l}
\quad 5 \\
\quad 5 \\
2 \longmapsto 2 \quad 5 \longmapsto 5 \\
2 \longmapsto 2 \quad 5 \longmapsto 5 \\
\quad f \qquad\quad g
\end{array}
&
\rightsquigarrow
&
\begin{array}{l}
\quad\; 5 \longmapsto 5 \\
25 \longmapsto 5 \longmapsto 5 \\
\quad\; 2 \longmapsto 2 \\
4 \longmapsto 2 \longmapsto 2 \\
\qquad f \otimes g
\end{array}
\end{array}
$$

We prove that the logical products in **Nest** are compatible with logical products of layouts.

Theorem F. *(see 3.2.6.31) If f and g are non-degenerate **Nest**-morphisms and f and g are product admissible, then*

$$L_{f \otimes g} = L_f \otimes L_g.$$
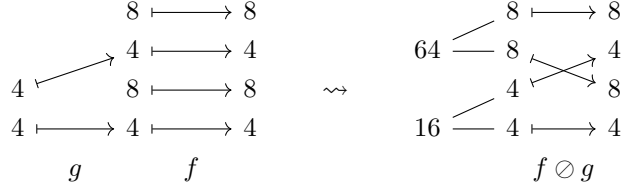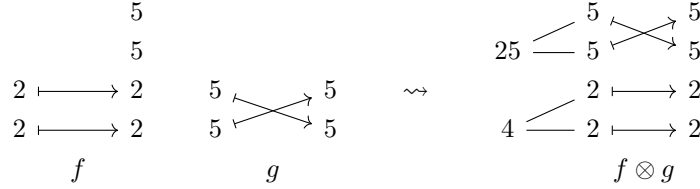
In Chapter 4, we illustrate how our new framework may be used to compute important layout operations such as composition, logical division, and logical products. In particular, we present an algorithm (Algorithm 4.1.3) for computing the composition $B \circ A$ of tractable layouts A and B. Eliding details, the basic idea of our algorithm is that if we want to compute the composition $B \circ A$, we can represent A and B by suitably chosen **Nest**-morphisms f and g, compose these morphisms to form $g \circ f$, then take the encoded layout to obtain

$$B \circ A = L_{g \circ f}.$$

We illustrate this algorithm with many examples.

1.2 Organization

The current work is organized as follows.

In section 1.3, we provide details regarding the `cute` implementation of layouts. We provide a Python implementation of the category **Nest** in the form of a module `tract`, and illustrate the compatibility of `tract` with `cute`. Our Python implementation may be found at our git repository https://github.com/ColfaxResearch/layout-categories.

Chapter 2 serves as a comprehensive reference for layouts and their algebra. It provides rigorous definitions of layouts and the operations they support, and establishes the fundamental properties of these operations. This chapter is replete with examples, and may be of use to the working programmer.

In Chapter 3, we present a new mathematical framework for working with tractable layouts. In particular, we connect layouts and their algebra to the theory of **categories** and **operads**. The content of this chapter is of independent mathematical interest. It is also of practical value, as it provides a new framework for visualizing layouts and computing their various operations.

In Chapter 4, we provide an algorithm for computing the composite of tractable layouts A and B using the framework developed in Chapter 3. We illustrate the composition algorithm with many examples.

1.3 Implementation

In this section, we illustrate how to work with layouts in NVIDIA's CuTe DSL, which we denote as `cute`. We provide an implementation of our categorical framework in the form of a `Python` module `tract` in our git repository https://github.com/ColfaxResearch/layout-categories. Here, we show the compatibility of `cute` and `tract`.

1. **Constructing tuples and nested tuples:** We construct tuples and nested tuples in `Python` as follows.

```
1  S = (2,2,2)
2  T = ((2,2),(5,5))
3  U = ((2,2),4,(9,(3,3)))
```

   Note that if we want to construct a tuple of length 1, we must include a comma following the tuple's entry. For example,

```
1  S = (10,)
2  T = (10)
```

   returns

```
1  S = (10,)
2  T = 10
```

2. **Constucting layouts and morphisms:** We construct a layout

$$L = S : D$$

   in `cute` as follows.

```
1  L = cute.make_layout(shape=S, stride=D)
```

   For example,

```
1  A = cute.make_layout(shape=((4,4),4), stride=((16,1),4))
2  B = cute.make_layout(shape=(8,64), stride=(64,1))
3  C = cute.make_layout(shape=100, stride=2)
```

   returns

```
1  A = ((4,4),4):((16,1),4)
2  B = (8,64):(64,1)
3  C = 100:2
```

   We construct a nested tuple morphism

$$S \xrightarrow[\alpha]{f} T$$

   in `tract` as follows.

```
1  f = tract.make_morphism(domain=S, codomain=T, map_=alpha)
```

   For example,

```
1  f = tract.make_morphism(domain=(4,4), codomain=(4,2,4), map_=(1,3))
2  g = tract.make_morphism(domain=(2,2,2,2), codomain=(2,2,2,2), map_=(1,0,4,2))
3  h = tract.make_morphism(domain=(16,(4,4),(4,4)), codomain=(16,4,4), map_=(1,2,0,3,0))
```

returns

```
1  f = (4,4)--(1,3)-->(4,2,4)
2  g = (2,2,2,2)--(1,0,4,2)-->(2,2,2,2)
3  h = (16,(4,4),(4,4))--(1,2,0,3,0)-->(16,4,4)
```

Note that we use the symbol 0 rather than $*$ when specifying maps in `tract`.

3. **Translating between tractable layouts and morphisms:** If L is a layout, we can check if L is tractable with

```
1  tract.is_tractable(L)
```

For example,

```
1  A = cute.make_layout(shape=(2,2,2), stride=(1,2,4))
2  B = cute.make_layout(shape=(2,2,2), stride=(1,7,4))
3  A_is_tractable = tract.is_tractable(A)
4  B_is_tractable = tract.is_tractable(B)
```

returns

```
1  A = (2,2,2):(1,2,4)
2  B = (2,2,2):(1,7,4)
3  A_is_tractable = True
4  B_is_tractable = False
```

If L is a tractable layout, then we can construct the standard representation f_L with

```
1  tract.compute_morphism(L)
```

For example,

```
1  L = cute.make_layout(shape=(2,2,2), stride=(1,2,4))
2  f_L = tract.compute_morphism(L)
```

returns

```
1  L = (2,2,2):(1,2,4)
2  f_L = (2,2,2)--(1,2,3)-->(2,2,2)
```

If f is a nested tuple morphism, we can construct the layout L_f encoded by f with

```
1  tract.compute_layout(f)
```

For example,

```
1  f = tract.make_morphism(domain=((5,5),8), codomain=(5,8,5), map_=(1,3,2))
2  L_f = tract.compute_layout(f)
```

returns

11

```
1  f = ((5,5),8)--(1,3,2)-->(5,8,5)
2  L_f = ((5,5),8):((1,40),5)
```

4. **Composition**: When defined, this operation produces a layout $B \circ A$ from a pair of layouts A and B. See Definition 2.3.7.1 for a precise definition. We can compute the composition $B \circ A$ in `cute` with

```
1  cute.composition(B,A)
```

For example, running

```
1  A = cute.make_layout(shape=((4,4),4), stride=((16,1),4))
2  B = cute.make_layout(shape=(8,64), stride=(64,1))
3  B_o_A = cute.composition(B,A)
```

returns

```
1  A = ((4,4),4):((16,1),4)
2  B = (8,64):(64,1)
3  B_o_A = ((4,4),(2,2)):((2,64),(256,1))
```

If f and g are composable nested tuple morphisms, we can compute the composition $g \circ f$ in `tract` with

```
1  tract.compose(f,g)
```

For example,

```
1  f = tract.make_morphism(domain=((2,2),(2,2)), codomain=((2,2,2),(2,2,2)), map_=(3,2,6,5))
2  g = tract.make_morphism(domain=((2,2,2),(2,2,2)), codomain=(2,2,2,2), map_=(1,0,2,0,3,4))
3  g_o_f = tract.compose(f,g)
```

returns

```
1  f = ((2,2),(2,2))--(3,2,6,5)-->((2,2,2),(2,2,2))
2  g = ((2,2,2),(2,2,2))--(1,0,2,0,3,4)-->(2,2,2,2)
3  g_o_f = ((2,2),(2,2))--(2,0,4,3)-->(2,2,2,2)
```

5. **Coalesce**: This operation produces a layout $\mathsf{coal}(A)$ from a layout A. See Definition 2.3.3.1 for details. We can compute $\mathsf{coal}(A)$ in `cute` with

```
1  cute.coalesce(A)
```

For example,

```
1  A = cute.make_layout(shape = ((2,2),(2,2),(5,5)), stride = ((1,2),(16,32),(64,640)))
2  coal_A = cute.coalesce(A)
```

returns

```
1  A = ((2,2),(2,2),(5,5)):((1,2),(16,32),(64,640))
2  coal_A = (4,20,5):(1,16,640)
```

There is also a *relative coalesce* operation $A \mapsto \mathsf{coal}(A, S)$, which receives as input an additional nested tuple S which is *refined* by the shape of A. See Definition 2.3.4.7 for details. We can compute $\mathsf{coal}(A, S)$ in `cute` with

```
A = cute.make_layout(shape = ((2,2),(3,3),(5,5)), stride = ((1,2),(4,12),(36,180)))
S = ((2,2),9,25)
coal_A_over_S = cute.coalesce(A,target_profile=S)
```

returns

```
A = ((2,2),(3,3),(5,5)):((1,2),(4,12),(36,180))
S = ((2,2),9,25)
coal_A_over_S = ((2,2),9,25):((1,2),4,36)
```

If f is a nested tuple morphism, we may form $\mathsf{coal}(f)$. See Definition 3.2.6.11 for details. We compute $\mathsf{coal}(f)$ in `tract` with

```
tract.coalesce(f)
```

For example,

```
f = tract.make_morphism(domain=(2,2,10,10), codomain = (2,2,2,10,10), map_=(1,2,4,5))
coal_f = tract.coalesce(f)
```

returns

```
f = (2,2,10,10)--(1,2,4,5)-->(2,2,2,10,10)
coal_f = (4,100)--(1,3)-->(4,2,100)
```

6. **Complement**: When defined, this operation produces a layout $\mathsf{comp}(A, N)$ from a layout A and positive integer N. See Definition 2.3.6.5 for details. We can compute $\mathsf{comp}(A, N)$ in `cute` with

```
cute.complement(A,N)
```

For example,

```
A = cute.make_layout(shape = ((2,2),(2,2)), stride = ((8,2),(64,256)))
comp_A = cute.complement(A,4096)
```

returns

```
A = ((2,2),(2,2)):((8,2),(64,256))
comp_A = (2,2,4,2,8):(1,4,16,128,512)
```

If f is a nested tuple morphism, then we may form the complement f^c of f. See Definition 3.2.6.17 for details. We compute f^c in `tract` with

```
tract.complement(f)
```

For example,

```
f = tract.make_morphism(domain=(2,2), codomain=(2,5,2,5), map_=(1,3))
comp_f = tract.complement(f)
```

returns

```
1  f = (2,2)--(1,3)-->(2,5,2,5)
2  comp_A = (5,5)--(2,4)-->(2,5,2,5)
```

7. **Logical Division**: When defined, this operation produces a layout $A \oslash B$ from a pair of layouts A and B. See Definition 2.3.8.1 for details. We compute $A \oslash B$ in cute with

```
1  cute.logical_divide(A,B)
```

For example,

```
1  A = cute.make_layout((64,32), stride = (32,1))
2  B = cute.make_layout((4,4), stride = (1,64))
3  quotient = cute.logical_divide(A,B)
```

returns

```
1  A = (64,32):(32,1)
2  B = (4,4):(1,64)
3  quotient = ((4,4),(16,8)):((32,1),(128,4))
```

If f and g are nested tuple morphisms and g divides f, then we may form the logical division $f \oslash g$. See Definition 3.2.6.23 for details. We compute $f \oslash g$ in tract with

```
1  tract.logical_divide(f,g)
```

For example,

```
1  f = tract.make_morphism(domain=(4,8,4,8), codomain=(4,8,4,8), map_=(1,2,3,4))
2  g = tract.make_morphism(domain=(4,4), codomain=(4,8,4,8), map_=(1,3))
3  quotient = tract.logical_divide(f,g)
```

returns

```
1  f = (4,8,4,8)--(1,2,3,4)-->(4,8,4,8)
2  g = (4,4)--(1,3)-->(4,8,4,8)
3  quotient = ((4,4),(8,8))--(1,3,2,4)-->(4,8,4,8)
```

8. **Logical Product**: When defined, this operation produces a layout $A \otimes B$ from a pair of layouts A and B. See Definition 2.3.9.1 for details. We compute $A \otimes B$ in cute with

```
1  cute.logical_product(A,B)
```

For example, running

```
1  A = cute.make_layout((3,10,10), stride = (200,1,20))
2  B = cute.make_layout((2,2), stride = (1,2))
3  product = cute.logical_product(A,B)
```

returns

```
1  A = (3,10,10):(200,1,20)
2  B = (2,2):(1,2)
3  product = ((3,10,10),(2,2)):((200,1,20),(10,600))
```

If f and g are nested tuple morphisms and f and g are product admissible, then we may form the logical product $f \otimes g$. See Definition 3.2.6.28 for details. We compute $f \otimes g$ in `tract` with

```
tract.logical_product(f,g)
```

For example,

```
f = tract.make_morphism(domain=(2,2), codomain=(2,2,5,5), map_=(1,2))
g = tract.make_morphism(domain=(5,5), codomain=(5,5), map_=(2,1))
product = tract.logical_product(f,g)
```

returns

```
f = (2,2)--(1,2)-->(2,2,5,5)
g = (5,5)--(2,1)-->(5,5)
product = ((2,2),(5,5))--(1,2,4,3)-->(2,2,5,5)
```

1.4 Notation

$$\mathbb{Z} = \{\dots, -1, 0, 1, 2, \dots\}$$

$$\mathbb{N} = \{0, 1, 2, \dots\}$$

$$\mathbb{Z}_{>0} = \{1, 2, \dots\}$$

$$[0, n) = \{0, \dots, n-1\}, \text{ and } [0, 0) = \varnothing.$$

$$\langle n \rangle = \{1, 2, \dots, n\}, \text{ and } \langle 0 \rangle = \varnothing.$$

$$\langle n \rangle_* = \{*, 1, 2, \dots, n\}$$

$$\delta_i^m = (0, \dots, 1, \dots, 0), \text{ the tuple of length } m \text{ with}$$
$$i\text{th entry } 1 \text{ and all other entries } 0.$$

$$\Sigma_n = \text{ the symmetric group on } \langle n \rangle.$$

$$X^\sigma = (x_{\sigma(1)}, \dots, x_{\sigma(m)}) \text{ for a tuple } X = (x_1, \dots, x_m)$$
$$\text{and a permutation } \sigma \in \Sigma_m.$$

$$X \star Y = \text{ the flat concatenation of } X \text{ and } Y.$$

$$X^\flat = \text{ the flattening of a nested tuple } X.$$

$$\mathsf{prof}(X) = \text{ the profile of a nested tuple } X.$$

$$(X_1, \dots, X_k) = \text{ the (nested) concatenation of } X_1, \dots, X_k.$$

$$(X_1, \dots, X_k)_Q = \text{ the } Q\text{-substitution of } X_1, \dots, X_k \text{ for a profile } Q.$$

$$\mathsf{Tuple}(V) = \text{ the set of tuples with entries in a set } V.$$

$$\mathsf{Nest}(V) = \text{ the set of nested tuples with entries in a set } V.$$

$$\mathsf{Profile} = \text{ the set of profiles.}$$

$$\mathsf{FlatLayout} = \text{ the set of flat layouts.}$$

$$\mathsf{Layout} = \text{ the set of layouts.}$$

$$B \circ A = \text{ the composition of } A \text{ and } B.$$

$$A \oslash B = \text{ the logical division of } A \text{ by } B.$$

$$A \otimes B = \text{ the logical product of } A \text{ and } B.$$

$$\mathbf{Set} = \text{ the category of sets.}$$

$$\mathbf{FinSet} = \text{ the category of finite sets.}$$

$$\mathbf{Fin} = \text{ the full subcategory of } \mathbf{FinSet} \text{ spanned by } \langle n \rangle \text{ for } n \geq 0.$$

$$\mathbf{FinSet}_* = \text{ the category of pointed finite sets.}$$

$$\mathbf{Fin}_* = \text{ the full subcategory of } \mathbf{FinSet}_* \text{ spanned by } \langle n \rangle_* \text{ for } n \geq 0.$$

$$\mathbf{Tuple} = \text{ the category of tuples and tuple morphisms.}$$

$$\mathbf{Nest} = \text{ the category of nested tuples and nested tuple morphisms.}$$

$$\mathbf{Ref} = \text{ the category of nested tuples and refinements.}$$

$$\mathbf{Cat} = \text{ the category of (small) categories and functors.}$$

Chapter 2

Layouts and their algebra

The goal of this chapter is to provide a comprehensive and mathematically grounded theory of layouts. We begin by developing a theory of flat layouts in section 2.1. We introduce the necessary background on nested tuples in section 2.2, so that we may cover layouts in full generality in section 2.3.

2.1 Flat Layouts

In this section, we examine **flat layouts**, an important subclass of layouts in which both shape and stride are tuples, rather than more general nested tuples. To formalize our discussion, we begin by fixing notation related to tuples.

2.1.1 Tuples

Definition 2.1.1.1. If V is a set, then a *tuple* with entries in V is a finite ordered list

$$X = (x_1, \ldots, x_m)$$

of elements $x_i \in V$ for each $1 \leq i \leq m$. The *length* of such a tuple $X = (x_1, \ldots, x_m)$ is

$$\mathsf{len}(X) = m.$$

We write $\mathsf{Tuple}(V)$ for the collection of all tuples with entries in V. We are especially interested in the case $V = \mathbb{Z}$, in which case we refer to $X \in \mathsf{Tuple}(\mathbb{Z})$ as a *tuple of integers*. If X is a tuple of integers, then the *size* of X is the product

$$\mathsf{size}(X) = x_1 \cdots x_m.$$

Example 2.1.1.2. Here are some examples of tuples, together with their length and size:

$$\begin{aligned}
X &= (3, 128, 128), & \mathsf{len}(X) &= 3, & \mathsf{size}(X) &= 49152 \\
X &= (512), & \mathsf{len}(X) &= 1, & \mathsf{size}(X) &= 512 \\
X &= (), & \mathsf{len}(X) &= 0, & \mathsf{size}(X) &= 1
\end{aligned}$$

Definition 2.1.1.3. If $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$ are tuples, then we write

$$X \star Y = (x_1, \ldots, x_m, y_1, \ldots, y_n)$$

for the *concatenation* of X and Y.

Example 2.1.1.4. If $X = (64, 32)$ and $Y = (8, 8, 8)$, then

$$X \star Y = (64, 32, 8, 8, 8).$$

Remark 2.1.1.5. If V is a set, then the collection

$$\mathsf{Tuple}(V) = \coprod_{m \geq 0} V^{\times m}$$

of all tuples with entries in V is the free associative monoid on V. The monoidal product is concatenation, and the unit is the empty tuple $()$.

Definition 2.1.1.6. If X and X' are tuples, we say X' *divides* X if there exists a tuple X'' with

$$X' \star X'' = X.$$

Example 2.1.1.7. If $X' = (81, 9)$ and $X = (81, 9, 64, 8)$, then X' divides X, since the tuple $X'' = (64, 8)$ satisfies

$$X' \star X'' = X.$$

Definition 2.1.1.8. If $X = (x_1, \ldots, x_m)$ is a tuple and $\sigma \in \Sigma_m$ is a permutation, then we write

$$X^\sigma = (x_{\sigma(1)}, \ldots, x_{\sigma(m)})$$

for the *permutation of X by σ*. This specifies a right action of Σ_m on $\mathbb{Z}^{\times m}$.

Example 2.1.1.9. If $X = (8, 16, 32, 64)$ and $\sigma = (1\ 2)(3\ 4)$, then

$$X^\sigma = (16, 8, 64, 32).$$

Notation 2.1.1.10. If n is a positive integer, we write

$$[0, n) = \{0, 1, \ldots, n - 1\},$$

and if $S = (s_1, \ldots, s_m)$ is a tuple of positive integers, we write

$$[0, S) = [0, s_1) \times \cdots \times [0, s_m)$$

for the collection of tuples $(x_1, \ldots, x_m)$ with $0 \leq x_i < s_i$.

Example 2.1.1.11. If $S = (3, 2)$, then

$$[0, S) = \{(0, 0), (1, 0), (2, 0), (0, 1), (1, 1), (2, 1)\}$$

2.1.2 Basic definitions

Having fixed notation, we are ready to define flat layouts.

Definition 2.1.2.1. A *flat layout* is a pair

$$L = S : D$$
$$= (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

18

consisting of a tuple of positive integers

$$\mathsf{shape}(L) = S$$
$$= (s_1, \ldots, s_m)$$

called the *shape* of L, and a tuple of non-negative integers

$$\mathsf{stride}(L) = D$$
$$= (d_1, \ldots, d_m)$$

called the *stride* of L.

Remark 2.1.2.2. If L is a flat layout, then by definition, $\mathsf{shape}(L)$ and $\mathsf{stride}(L)$ have the same length.

Remark 2.1.2.3. A flat layout is an example of the more general *layout* of Definition 2.3.1.1, so we sometimes refer to a flat layout L as a *layout*.

Example 2.1.2.4. Here are some examples of flat layouts:

$$L_1 = (2, 2, 2) : (1, 2, 4),$$
$$L_2 = (128) : (5),$$
$$L_3 = (16, 12, 512, 512) : (0, 0, 1, 512),$$
$$L_4 = (6, 1, 12, 2, 2) : (2, 0, 12, 144, 1),$$
$$L_5 = () : ().$$

Example 2.1.2.5. We can depict the layout $L = (8) : (5)$ as

$L =$	0	5	10	15	20	25	30	35

and we can depict the layout $L = (3, 5) : (2, 10)$ as

$L =$	0	10	20	30	40
	2	12	22	32	42
	4	14	24	34	44

We make precise the sense in which these pictures represent the associated layout in Remark 2.1.2.17.

Perhaps the most important examples of flat layouts are the column-major and row-major layouts, which we define below.

Definition 2.1.2.6. Suppose

$$L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

is a flat layout. We say L is *column-major* if

$$d_i = s_1 \cdots s_{i-1}$$

for each $1 \le i \le m$. We say L is *row-major* if

$$d_i = s_{i+1} \cdots s_m.$$

for each $1 \le i \le m$.

Example 2.1.2.7. The layout

$$L = (3,4) : (1,3) =$$

0	3	6	9
1	4	7	10
2	5	8	11

is column-major, while the layout

$$L = (3,4) : (4,1) =$$

0	1	2	3
4	5	6	7
8	9	10	11

is row-major. These pictures make clear the reason for the terminology: If L is a column-major layout of rank 2, then the columns of L are contiguous, and if L is a row-major layout of rank 2, then the rows of L are contiguous.

Example 2.1.2.8. The layouts

$$L_1 = (2,2,2,2,2) : (1,2,4,8,16)$$
$$L_2 = (3,128,128) : (1,3,384)$$
$$L_3 = (64) : (1)$$

are column-major, while the layouts

$$L_4 = (2,2,2,2,2) : (16,8,4,2,1)$$
$$L_5 = (3,128,128) : (16384,128,1)$$
$$L_6 = (64) : (1)$$

are row-major.

Now that we've seen a few examples, lets define some important attributes of flat layouts.

Definition 2.1.2.9. Suppose $L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$ is a flat layout.

- The *rank* of L is
$$\mathsf{rank}(L) = m.$$

20

- The *size* of L is

$$\mathsf{size}(L) = \prod_{i=1}^{m} s_i.$$

- The *cosize* of L is

$$\mathsf{cosize}(L) = 1 + \sum_{i=1}^{m} (s_i - 1) \cdot d_i.$$

- For any $1 \leq i \leq \mathsf{rank}(L)$, the *ith mode* of L is the pair

$$\mathsf{mode}_i(L) = s_i : d_i.$$

Example 2.1.2.10. The layout

$$L = (64, 32) : (1, 128)$$

has $\mathsf{rank}(L) = 2$, $\mathsf{size}(L) = 2048$, and $\mathsf{cosize}(L) = 4032$. The modes of L are

$$\mathsf{mode}_1(L) = 64 : 1$$
$$\mathsf{mode}_2(L) = 32 : 128.$$

Example 2.1.2.11. The layout

$$L = (3, 8, 8, 8) : (1, 3, 24, 192).$$

has $\mathsf{rank}(L) = 4$, $\mathsf{size}(L) = 1536$, and $\mathsf{cosize}(L) = 1536$. The layout L has four modes, for example $\mathsf{mode}_3(L) = 8 : 24$.

Example 2.1.2.12. The layout

$$L = (2, 2, 2, 2, 2) : (160, 80, 40, 20, 10).$$

has $\mathsf{rank}(L) = 5$, $\mathsf{size}(L) = 32$, and $\mathsf{cosize}(L) = 311$. The layout L has 5 modes, for example $\mathsf{mode}_5(L) = 2 : 10$.

If L is a flat layout, then L encodes a *coordinate function* φ_L. The coordinate function of L is a multi-dimensional to one-dimensional transformation given by taking a dot product with $\mathsf{stride}(L)$. Recall that if $S = (s_1, \ldots, s_m)$ is a tuple of positive integers, then

$$[0, S) = [0, s_1) \times \cdots \times [0, s_m)$$

is the set of all tuples $(x_1, \ldots, x_m)$ such that $0 \leq x_i < s_i$. In particular, if $S = ()$ is the empty tuple, then $[0, S) = \{()\}$.

Construction 2.1.2.13 (Coordinate functions)**.** If

$$L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

is a flat layout, then the *coordinate function* of L is the function

$$[0, \mathsf{shape}(L)) \xrightarrow{\varphi_L} \mathbb{Z}$$

given by

$$\varphi_L(x_1, \ldots, x_m) = (x_1, \ldots, x_m) \cdot (d_1, \ldots, d_m)$$
$$= x_1 d_1 + \cdots + x_m d_m.$$

21

The coordinate function φ_L factors through the inclusion $[0, \mathsf{cosize}(L)) \subset \mathbb{Z}$, and we write

$$[0, \mathsf{shape}(L)) \xrightarrow{\;\varphi_L^{\mathsf{cosize}(L)}\;} [0, \mathsf{cosize}(L)) \subset \mathbb{Z}$$

for the factored map. More generally, for any $N \geq \mathsf{cosize}(L)$, we write φ_L^N for the factorization of φ_L through $[0, N) \subset \mathbb{Z}$, and by a mild abuse of terminology, we refer to such a map φ_L^N as the *coordinate function* of L.

Example 2.1.2.14. If $L = (2,3) : (1,5)$, then the coordinate function

$$\varphi_L : [0,2) \times [0,3) \to \mathbb{Z}$$

is given by

$$\begin{aligned}
\varphi_L(0,0) &= (0,0) \cdot (1,5) = 0, \\
\varphi_L(1,0) &= (1,0) \cdot (1,5) = 1, \\
\varphi_L(0,1) &= (0,1) \cdot (1,5) = 5, \\
\varphi_L(1,1) &= (1,1) \cdot (1,5) = 6, \\
\varphi_L(0,2) &= (0,2) \cdot (1,5) = 10, \\
\varphi_L(1,2) &= (1,2) \cdot (1,5) = 11.
\end{aligned}$$

Example 2.1.2.15. If $L = (2,2) : (64,2)$, then the coordinate function

$$\varphi_L : [0,2) \times [0,2) \to \mathbb{Z}$$

is given by

$$\begin{aligned}
\varphi_L(0,0) &= (0,0) \cdot (64,2) = 0, \\
\varphi_L(1,0) &= (1,0) \cdot (64,2) = 64, \\
\varphi_L(0,1) &= (0,1) \cdot (64,2) = 2, \\
\varphi_L(1,1) &= (1,1) \cdot (64,2) = 66.
\end{aligned}$$

Example 2.1.2.16. If $E = () : ()$ is the empty layout, then the coordinate function of E is the map

$$\varphi_E : \{()\} \to \mathbb{Z}$$

given by

$$\varphi(()) = 0.$$

Remark 2.1.2.17. We can now, for example, give a precise description of the sense in which the image

0	10	20	30	40
2	12	22	32	42
4	14	24	34	44

depicts the layout $L = (3, 5) : (2, 10)$: The (i, j)th cell of the grid is labeled by the value

$$\varphi_L(i, j) = (i, j) \cdot (2, 10) = 2i + 10j$$

of the coordinate function of L.

In practice, the most important invariant of a flat layout L is its *layout function* Φ_L, which is obtained by precomposing the coordinate function

$$\varphi_L : [0, S) \to \mathbb{Z}$$

with the inverse of the *colexicographic isomorphism*

$$\mathsf{colex}_S : [0, S) \to [0, \mathsf{size}(S)).$$

Definition 2.1.2.18. Suppose $S = (s_1, \ldots, s_m)$ is a tuple of positive integers and recall that

$$[0, S) = [0, s_1) \times \cdots \times [0, s_m).$$

The *colexicographic isorphism* is the map

$$[0, S) \xrightarrow{\quad \mathsf{colex}_S \quad} [0, \mathsf{size}(S))$$

$$(x_1, \ldots, x_m) \longmapsto \sum_{i=1}^{m} s_1 \cdots s_{i-1} x_i.$$

We sometimes write $\mathsf{colex} = \mathsf{colex}_S$ when the tuple S is clear from context. The inverse of the colexicographic isomorphism is the map

$$[0, \mathsf{size}(S)) \xrightarrow{\quad \mathsf{colex}_S^{-1} \quad} [0, S)$$

given by

$$\mathsf{colex}_S^{-1}(x) = (x_1, \ldots, x_m)$$

where

$$x_i = \left\lfloor \frac{x}{s_1 \cdots s_{i-1}} \right\rfloor \mod s_i.$$

Note that if $S = ()$ is the empty tuple, then

$$\mathsf{colex}_{()} : \{()\} \to \{0\}$$

and

$$\mathsf{colex}_{()}^{-1} : \{0\} \to \{()\}$$

are the canonical isomorphisms.

Construction 2.1.2.19 (Layout functions)**.** If

$$L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m),$$

is a flat layout, then the *layout function of L* is the composite

$$[0, \mathsf{size}(L)) \xrightarrow{\quad\quad \Phi_L \quad\quad} \mathbb{Z}.$$

$$\mathsf{colex}_{\mathsf{shape}(L)}^{-1} \searrow \quad \nearrow \varphi_L$$

$$[0, \mathsf{shape}(L))$$

23

Explicitly, Φ_L is given by

$$\Phi_L(x) = x_1 d_1 + \cdots + x_m d_m$$

where

$$x_i = \left\lfloor \frac{x}{s_1 \cdots s_{i-1}} \right\rfloor \mod s_i.$$

The layout function Φ_L factors through the inclusion $[0, \mathsf{cosize}(L)) \subset \mathbb{Z}$, and we write

$$[0, \mathsf{size}(L)) \xrightarrow{\;\;\Phi_L^{\mathsf{cosize}(L)}\;\;} [0, \mathsf{cosize}(L)) \subset \mathbb{Z}$$

for the factored map. More generally, for any $N \geq \mathsf{cosize}(L)$, we write Φ_L^N for the factorization of Φ_L through $[0, N) \subset \mathbb{Z}$, and by a mild abuse of terminology, we refer to such a map φ_L^N as the *layout function* of L.

Example 2.1.2.20. If $L = (2, 3) : (1, 5)$, then the layout function

$$\Phi_L : [0, 6) \to \mathbb{Z}$$

is given by

$$\Phi_L(0) = (0, 0) \cdot (1, 5) = 0,$$
$$\Phi_L(1) = (1, 0) \cdot (1, 5) = 1,$$
$$\Phi_L(2) = (0, 1) \cdot (1, 5) = 5,$$
$$\Phi_L(3) = (1, 1) \cdot (1, 5) = 6,$$
$$\Phi_L(4) = (0, 2) \cdot (1, 5) = 10,$$
$$\Phi_L(5) = (1, 2) \cdot (1, 5) = 11.$$

Example 2.1.2.21. If $L = (2, 2) : (64, 2)$, then the layout function

$$\Phi_L : [0, 4) \to \mathbb{Z}$$

is given by

$$\Phi_L(0) = (0, 0) \cdot (64, 2) = 0,$$
$$\Phi_L(1) = (1, 0) \cdot (64, 2) = 64,$$
$$\Phi_L(2) = (0, 1) \cdot (64, 2) = 2,$$
$$\Phi_L(3) = (1, 1) \cdot (64, 2) = 66.$$

Example 2.1.2.22. If $L = (4, 2, 2) : (3, 3, 100)$, then for example, the layout function of L satisfies

$$\Phi_L(7) = (3, 1, 0) \cdot (3, 3, 100) = 12,$$
$$\Phi_L(9) = (1, 0, 1) \cdot (3, 3, 100) = 103.$$

Example 2.1.2.23. If $E = () : ()$ is the empty layout, then

$$\Phi_E : \{0\} \to \mathbb{Z}$$

is given by

$$\Phi_E(0) = 0.$$

24

Example 2.1.2.24. If L is any flat layout, then the layout function Φ_L of L satisfies

$$\Phi_L(0) = 0.$$

Remark 2.1.2.25. If $S = (s_1, \ldots, s_m)$ is a tuple of positive integers, then the colexicographic isomorphism

$$[0, S) \xrightarrow{\;\;\mathsf{colex}_S\;\;} [0, \mathsf{size}(S))$$

is equal to the coordinate function $\varphi_L^{\mathsf{cosize}(L)}$ of the column major layout

$$L = (s_1, s_2, \ldots, s_m) : (1, s_1, \ldots, s_1 \cdots s_{m-1}).$$

This implies that if a flat layout L is column-major, then

$$
\begin{aligned}
\Phi_L^{\mathsf{cosize}(L)} &= \varphi_L^{\mathsf{cosize}(L)} \circ \mathsf{colex}_{\mathsf{shape}(L)}^{-1} \\
&= \varphi_L^{\mathsf{cosize}(L)} \circ \left(\varphi_L^{\mathsf{cosize}(L)} \right)^{-1} \\
&= \mathsf{id}_{[0, \mathsf{size}(L))}
\end{aligned}
$$

is the identity map on $[0, \mathsf{size}(L))$.

Remark 2.1.2.26. There exist distinct layouts $A \neq B$ with $\Phi_A = \Phi_B$. For example, the layouts

$$
\begin{aligned}
A &= (7, 7) : (1, 7) \\
B &= (49) : (1)
\end{aligned}
$$

are not equal, yet $\Phi_A = \Phi_B$. Later, we will characterize precisely when two flat layouts A and B have the same layout function (see Proposition 2.1.4.18).

Before moving on to our discussion of layout operations, we need to define the notion of non-degeneracy.

Definition 2.1.2.27. Suppose

$$L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

is a flat layout. We say L is *non-degenerate* if for any $1 \leq i \leq m$, we have

$$s_i = 1 \quad \Rightarrow \quad d_i = 0.$$

Example 2.1.2.28. The layouts

$$
\begin{aligned}
L_1 &= (4, 1) : (1, 0) \\
L_2 &= (8, 1, 8, 1) : (2, 0, 16, 0)
\end{aligned}
$$

are non-degenerate, while the layouts

$$
\begin{aligned}
L_3 &= (4, 1) : (1, 4) \\
L_4 &= (8, 1, 8, 1) : (2, 16, 16, 256)
\end{aligned}
$$

are degenerate.

Observation 2.1.2.29. There is no real loss of generality in assuming that a layout L is non-degenerate. More precisely, if

$$
\begin{aligned}
L &= (s_1, \ldots, s_m) : (d_1, \ldots, d_m) \\
L' &= (s_1, \ldots, s_m) : (d_1', \ldots, d_m')
\end{aligned}
$$

are flat layouts with the same shape, and $d_i = d_i'$ whenever $s_i > 1$, then $\varphi_L = \varphi_{L'}$, and $\Phi_L = \Phi_{L'}$. In particular, we are free to set $d_i = 0$ whenever $s_i = 1$ without altering the coordinate function or layout function of L.

2.1.3 Basic operations

Having established the basic vocabulary for flat layouts, we turn to the operations they support. In this section, we define basic operations that will be needed to construct more sophisticated operations such as *coalesce*, *complement*, and *composition*.

2.1.3.1 Restriction

If L is a flat layout, it is often useful to restrict to a subset of the modes of L. Recall that for a non-negative integer m, we write
$$\langle m \rangle = \{1, \ldots, m\}.$$

Definition 2.1.3.1. Suppose
$$L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$
is a flat layout, and suppose
$$I = \{i_1 < \cdots < i_k\} \subset \langle m \rangle$$
is a subset. We define the *restriction of L to I* to be the flat layout
$$L \mid_I = (s_{i_1}, \ldots, s_{i_k}) : (d_{i_1}, \ldots, d_{i_k}).$$

Example 2.1.3.2. If

$$L = (3, 6) : (10, 5) =$$

0	5	10	15	20	25
10	15	20	25	30	35
20	25	30	35	40	45

and $I = \{2\}$, then

$$L \mid_I = (6) : (5) =$$

0	5	10	15	20	25

Example 2.1.3.3. If
$$L = (3, 8, 8, 8) : (1, 3, 24, 192)$$
and $I = \{1, 2, 3\}$, then
$$L \mid_I = (3, 8, 8) : (1, 3, 24).$$

Example 2.1.3.4. If
$$L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$
is a flat layout and $I = \langle m \rangle$, then
$$L \mid_I = L.$$

Example 2.1.3.5. If
$$L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$
is a flat layout and $I = \varnothing$ is the empty set, then
$$L \mid_I = () : ()$$
is the empty layout.

2.1.3.2 Squeeze

If L is a flat layout, then the operation $L \mapsto \mathsf{squeeze}(L)$ removes all modes $s_i : d_i$ of L where $s_i = 1$.

Construction 2.1.3.6. Suppose

$$L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

is a flat layout, and let

$$I = \{i \in \langle m \rangle \mid s_i > 1\}$$

be the collection of indices whose corresponding shape entry is not equal to 1. We define

$$\mathsf{squeeze}(L) = L \mid_I .$$

Example 2.1.3.7. If

$$L = (64, 64, 1) : (1, 64, 0),$$

then

$$\mathsf{squeeze}(L) = (64, 64) : (1, 64).$$

Example 2.1.3.8. If

$$L = (64, 64, 1, 32, 1) : (2048, 32, 0, 1, 0)$$

then

$$\mathsf{squeeze}(L) = (64, 64, 32) : (2048, 32, 1).$$

Example 2.1.3.9. If L is a flat layout, then

$$\mathsf{squeeze}(L) = L$$

if and only if $\mathsf{shape}(L)$ contains no entries equal to 1.

Example 2.1.3.10. If L is a flat layout, then

$$\mathsf{squeeze}(L) = (\,) : (\,)$$

is the empty layout if and only if all entries of $\mathsf{shape}(L)$ are equal to 1.

An essential property of this construction is that $L \mapsto \mathsf{squeeze}(L)$ leaves the layout function of L unchanged.

Lemma 2.1.3.11. *If L is a flat layout, then*

1. $\mathsf{size}(\mathsf{squeeze}(L)) = \mathsf{size}(L)$,

2. $\mathsf{cosize}(\mathsf{squeeze}(L)) = \mathsf{cosize}(L)$, *and*

3. $\Phi_{\mathsf{squeeze}(L)} = \Phi_L$.

Proof. Let

$$I = \{i_1 < \cdots < i_k\} \subset \langle m \rangle$$

denote the collection of indices with $s_{i_j} > 1$, so that

$$\mathsf{squeeze}(L) = (s_{i_1}, \ldots, s_{i_k}) : (d_{i_1}, \ldots, d_{i_k}).$$

For the first assertion, we compute

$$\mathsf{size}(\mathsf{squeeze}(L)) = \prod_{j=1}^{k} s_{i_j} = \left(\prod_{j=1}^{k} s_{i_j}\right) \cdot \left(\prod_{\langle m\rangle\setminus I} 1\right) = \prod_{i=1}^{m} s_i = \mathsf{size}(L).$$

For the second assertion, we compute

$$\mathsf{cosize}(\mathsf{squeeze}(L)) = 1 + \sum_{j=1}^{k}(s_{i_j} - 1) \cdot d_{i_j} = 1 + \sum_{j=1}^{k}(s_{i_j} - 1) \cdot d_{i_j} + \left(\sum_{\langle m\rangle\setminus I} 0\right)$$

$$= 1 + \sum_{i=1}^{m}(s_i - 1) \cdot d_i$$

$$= \mathsf{cosize}(L).$$

For the third assertion, it suffices to show that removing a mode of the form $1 : d_i$ from a flat layout leaves the layout function unchanged. Suppose $L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$, and suppose that some $s_i = 1$. Let

$$L' = (s'_1, \ldots, s'_{m-1}) : (d'_1, \ldots, d'_{m-1})$$

denote the flat layout obtained from L by removing its ith mode, so that

$$s'_j = \begin{cases} s_j & j < i \\ s_{j+1} & i \le j < m, \end{cases} \quad \text{and} \quad d'_j = \begin{cases} d_j & j < i \\ d_{j+1} & i \le j < m. \end{cases}$$

The layout function for L is given by

$$\Phi_L(x) = x_1 d_1 + \cdots + x_m d_m$$

where $x_j = \left\lfloor \dfrac{x}{s_1 \cdots s_{j-1}} \right\rfloor \mod s_j$, and the layout function for L' is given by

$$\Phi_{L'}(x) = x'_1 d'_1 + \cdots + x'_{m-1} d'_{m-1}$$

where $x'_j = \left\lfloor \dfrac{x}{s'_1 \cdots s'_{j-1}} \right\rfloor \mod s'_j$. We observe that

$$x'_j = \begin{cases} x_j & j < i \\ x_{j+1} & i \le j < m, \end{cases}$$

and since $x_i \in [0, 1)$ is necessarily 0, we have

$$\begin{aligned} \Phi_L(x) &= x_1 d_1 + \cdots + x_m d_m \\ &= x_1 d_1 + \cdots + x_{i-1} d_{i-1} + x_{i+1} d_{i+1} + \cdots + x_m d_m \\ &= x'_1 d'_1 + \cdots + x'_{m-1} d'_{m-1} \\ &= \Phi_{L'}(x). \end{aligned}$$

$\square$

2.1.3.3 Filter zeros

If L is a flat layout, then the operation $L \mapsto \mathsf{filter}(L)$ removes all modes $s_i : d_i$ with $d_i = 0$.

Definition 2.1.3.12. Suppose

$$L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

is a flat layout, and let

$$I = \{i \in \langle m \rangle \mid d_i > 0\}$$

be the collection of indices whose corresponding stride entry is not equal to 0. We define

$$\mathsf{filter}(L) = L \mid_I .$$

Example 2.1.3.13. If

$$L = (64, 8, 8, 128) : (8, 1, 0, 512)$$

then

$$\mathsf{filter}(L) = (64, 8, 128) : (8, 1, 512).$$

Example 2.1.3.14. If

$$L = (3, 2) : (12, 0) = \qquad \begin{array}{|c|c|} \hline 0 & 0 \\ \hline 12 & 12 \\ \hline 24 & 24 \\ \hline \end{array}$$

then

$$\mathsf{filter}(L) = (3) : (12) = \qquad \begin{array}{|c|} \hline 0 \\ \hline 12 \\ \hline 24 \\ \hline \end{array}$$

Example 2.1.3.15. If

$$L = (3, 8, 8, 8) : (16, 0, 0, 0)$$

then

$$\mathsf{filter}(L) = (3) : (16).$$

Example 2.1.3.16. If L is a flat layout, then

$$\mathsf{filter}(L) = L$$

if and only if all entries of $\mathsf{stride}(L)$ are nonzero.

Example 2.1.3.17. If L is a flat layout, then

$$\mathsf{filter}(L) = () : ()$$

is the empty layout if and only if all entries of $\mathsf{stride}(L)$ are equal to 0.

2.1.3.4   Permute

Recall that if $X = (x_1, \ldots, x_m)$ is a tuple of length m and $\sigma \in \Sigma_m$ is a permutation, then we write

$$X^\sigma = (x_{\sigma(1)}, \ldots, x_{\sigma(m)}).$$

for the permutation of X by σ.

Definition 2.1.3.18. If $L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$ is a flat layout of rank m and $\sigma \in \Sigma_m$ is a permutation, we define

$$\begin{aligned}
L^\sigma &= \mathsf{shape}(L)^\sigma : \mathsf{stride}(L)^\sigma \\
&= (s_{\sigma(1)}, \ldots, s_{\sigma(m)}) : (d_{\sigma(1)}, \ldots, d_{\sigma(m)}).
\end{aligned}$$

Example 2.1.3.19. If

$$L = (4, 2) : (12, 2) =$$

0	2
12	14
24	26
36	38

and $\sigma = (1\ 2) \in \Sigma_2$ is the transposition, then

$$L^\sigma = (2, 4) : (2, 12) =$$

0	12	24	36
2	14	26	38

is the transposed layout.

Example 2.1.3.20. If

$$L = (15, 12, 10) : (240, 1, 24)$$

and $\sigma = (1\ 2) \in \Sigma_3$, then

$$L^\sigma = (12, 15, 10) : (1, 240, 24).$$

Example 2.1.3.21. If

$$L = (2, 2, 2, 2, 2) : (1, 2, 4, 8, 16)$$

and $\sigma = (1\ 5)(3\ 2\ 4) \in \Sigma_5$, then

$$L^\sigma = (2, 2, 2, 2, 2) : (16, 8, 2, 4, 1).$$

Example 2.1.3.22. If

$$L = (s, \ldots, s) : (d, \ldots, d)$$

is a flat layout all of whose modes are equal, then for any $\sigma \in \Sigma_m$, we have

$$L^\sigma = L.$$

30

2.1.3.5 Sort

If L is a flat layout, it is often useful to permute L so that its modes are increasing, in the following sense.

Definition 2.1.3.23. We define a linear ordering on pairs $s : d$ of integers by

$$s : d \preceq s' : d' \quad \Leftrightarrow \quad \begin{aligned} &d < d', \text{ or} \\ &d = d' \text{ and } s \leq s'. \end{aligned}$$

Example 2.1.3.24. We have

$$5 : 8 \preceq 4 : 12 \preceq 5 : 12.$$

Definition 2.1.3.25. Suppose L is a flat layout. We say L is *sorted* if for any $1 \leq i < \mathsf{rank}(L)$, we have

$$\mathsf{mode}_i(L) \preceq \mathsf{mode}_{i+1}(L).$$

Example 2.1.3.26. The layouts

$$L_1 = (128, 64, 2, 2) : (1, 128, 8192, 16384)$$
$$L_2 = (2, 2, 2) : (1, 1, 1)$$

are sorted, while the layouts

$$L_3 = (2, 4, 8, 16) : (64, 1, 2, 4)$$
$$L_4 = (5, 32, 16) : (1, 5, 5)$$

are not sorted.

Example 2.1.3.27. The empty layout $E = () : ()$ is sorted.

Example 2.1.3.28. If

$$L = (s_1, \ldots, s_m) : (0, \ldots, 0)$$

is a flat layout with all entries of $\mathsf{stride}(L)$ equal to 0, then L is sorted if and only if

$$s_1 \leq s_2 \leq \cdots \leq s_m.$$

Whether or not a flat layout L is sorted is intimately related to the behavior of the layout function Φ_L of L, as described in the following lemma.

Lemma 2.1.3.29. *Suppose L is a flat layout. If Φ_L is non-decreasing, then L is sorted.*

Proof. We prove the contrapositive. Suppose that L is not sorted. We will show that there exists some $x \leq y$ in the domain of Φ_L with $\Phi_L(x) > \Phi_L(y)$. If there exists some $1 \leq i < m$ such that $d_i > d_{i+1}$, then we can let

$$x = \prod_{j<i} s_j, \quad \text{and} \quad y = \prod_{j<i+1} s_j,$$

in which case $x < y$, but

$$\begin{aligned}
\Phi_L(x) &= (0, \ldots, 1, 0, \ldots, 0) \cdot (d_1, \ldots, d_i, d_{i+1}, \ldots, d_m) \\
&= d_i \\
&> d_{i+1} \\
&= (0, \ldots, 0, 1, \ldots, 0) \cdot (d_1, \ldots, d_m) \\
&= \Phi_L(y).
\end{aligned}$$

31

On the other hand, if there exists some $1 \le i < m$ such that $d_i = d_{i+1}$ and $s_i > s_{i+1}$, we can set

$$x = (s_i - 1)\left(\prod_{j<i} s_j\right), \quad \text{and} \quad y = (s_{i+1} - 1)\left(\prod_{j<i+1} s_j\right),$$

in which case $x < y$, but

$$
\begin{aligned}
\Phi_L(x) &= (0, \ldots, s_i - 1, 0, \ldots, 0) \cdot (d_1, \ldots, d_i, d_{i+1}, \ldots, d_m) \\
&= (s_i - 1)d_i \\
&> (s_{i+1} - 1)d_i \\
&= (s_{i+1} - 1)d_{i+1} \\
&= (0, \ldots, 0, s_{i+1} - 1, \ldots, 0) \cdot (d_1, \ldots, d_m) \\
&= \Phi_L(y).
\end{aligned}
$$

We conclude that Φ_L is not non-decreasing. $\qquad\square$

Remark 2.1.3.30. The converse of the previous lemma is false. For example, the flat layout

$$L = (3, 5, 7) : (1, 1, 1)$$

is sorted, but

$$\Phi_L(7) = (0, 2, 0) \cdot (1, 1, 1) = 2$$

is strictly greater than

$$\Phi_L(16) = (0, 0, 1) \cdot (1, 1, 1) = 1.$$

If L is a flat layout, then we can permute the modes of L to obtain a sorted layout $\mathsf{sort}(L)$.

Construction 2.1.3.31. Suppose

$$L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

is a flat layout. Define a linear ordering $\preceq$ on $\langle m \rangle$ by $i \preceq j$ if

1. $\mathsf{mode}_i(L) \preceq \mathsf{mode}_j(L)$, and

2. if $\mathsf{mode}_i(L) = \mathsf{mode}_j(L)$ then $i \le j$.

Let $\sigma \in \Sigma_m$ be the permutation associated to the linear ordering $\preceq$ of $\langle m \rangle$. We define $\mathsf{sort}(L)$ to be permutation of L by σ:

$$\mathsf{sort}(L) = L^\sigma.$$

Example 2.1.3.32. If

$$L = (2, 4, 8, 16) : (64, 1, 2, 4)$$

then

$$\mathsf{sort}(L) = (4, 8, 16, 2) : (1, 2, 4, 64).$$

Example 2.1.3.33. If

$$L = (5, 32, 16) : (1, 5, 5)$$

then

$$\mathsf{sort}(L) = (5, 16, 32) : (1, 5, 5).$$

Example 2.1.3.34. If L is sorted, then $\mathsf{sort}(L) = L$. In particular, this implies that $\mathsf{sort}(-)$ is an idempotent operation:
$$\mathsf{sort}(\mathsf{sort}(L)) = \mathsf{sort}(L).$$

Observation 2.1.3.35. If L is a flat layout, then typically $\Phi_{\mathsf{sort}(L)} \neq \Phi_L$. However, the layout functions Φ_L and $\Phi_{\mathsf{sort}(L)}$ always have the same image. To see this, let's write

$$L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m), \text{ and}$$
$$\mathsf{sort}(L) = (s_{\sigma(1)}, \ldots, s_{\sigma(m)}) : (d_{\sigma(1)}, \ldots, d_{\sigma(m)})$$

for some permutation $\sigma \in \Sigma_m$. If an integer n is in the image of Φ_L, then there exists a tuple $(x_1, \ldots, x_m) \in \prod_{i=1}^m [0, s_i)$ such that

$$x_1 d_1 + \cdots + x_m d_m = n$$

in which case the tuple $(x_{\sigma(1)}, \ldots, x_{\sigma(m)}) \in \prod_{i=1}^m [0, s_{\sigma(i)})$ satisfies

$$x_{\sigma(1)} d_{\sigma(1)} + \cdots + x_{\sigma(m)} d_{\sigma(m)} = n.$$

This proves that $\mathsf{Image}(\Phi_{\mathsf{sort}}(L)) \subseteq \mathsf{Image}(\Phi_L)$, and the reverse inclusion is proved similarly.

2.1.3.6 Concatenate

Recall that if $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$ are tuples, then the concatenation of X and Y is the tuple
$$X \star Y = (x_1, \ldots, x_m, y_1, \ldots, y_n).$$

This definition extends naturally to the concatenation of flat layouts.

Definition 2.1.3.36. Suppose

$$L_1 = S_1 : D_1$$
$$L_2 = S_2 : D_2$$

are flat layouts. Then the *concatenation* of L_1 and L_2 is the flat layout

$$L_1 \star L_2 = S_1 \star S_2 : D_1 \star D_2.$$

Concatenation of flat layouts is associative, so more generally, if $L_1, \ldots, L_k$ are flat layouts, we may form the concatenation

$$L_1 \star \cdots \star L_k.$$

Example 2.1.3.37. If $L_1 = (7, 2) : (2, 1)$ and $L_2 = (3, 3, 3) : (0, 10, 30)$, then

$$L_1 \star L_2 = (7, 2, 3, 3, 3) : (2, 1, 0, 10, 30).$$

Example 2.1.3.38. If $E = () : ()$ is the empty layout, then for any flat layout L we have

$$L \star E = L = E \star L.$$

Observation 2.1.3.39. Suppose
$$L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$
is a flat layout. If we write
$$L_i = (s_i) : (d_i),$$
then we can write L as the concatenation
$$L = L_1 \star \cdots \star L_m.$$

If $L_1, \ldots, L_k$ are flat layouts, then the layout function of the concatenation $L_1 \star \cdots \star L_k$ is determined by the layout functions of $L_1, \ldots, L_k$ as follows.

Proposition 2.1.3.40. *Suppose $L_1, \ldots, L_k$ are flat layouts of shape $S_1, \ldots, S_k$, and size $N_1, \ldots, N_k$, respectively. Then the coordinate function*
$$[0, S_1 \star \cdots \star S_k) \xrightarrow{\varphi_{L_1 \star \cdots \star L_k}} \mathbb{Z}$$
of $L_1 \star \cdots \star L_k$ is equal to the composite
$$[0, S_1 \star \cdots \star S_k) \xrightarrow{\cong} [0, S_1) \times \cdots \times [0, S_k) \xrightarrow{\varphi_{L_1} + \cdots + \varphi_{L_k}} \mathbb{Z},$$

$$X_1 \star \cdots \star X_k \longleftrightarrow (X_1, \ldots, X_k)$$

and the layout function
$$[0, N_1 \cdots N_k) \xrightarrow{\Phi_{L_1 \star \cdots \star L_k}} \mathbb{Z}$$
of $L_1 \star \cdots \star L_k$ is equal to the composite
$$[0, N_1 \cdots N_k) \xrightarrow{\mathrm{colex}^{-1}_{(N_1, \ldots, N_k)}} [0, N_1) \times \cdots \times [0, N_k) \xrightarrow{\Phi_{L_1} + \cdots + \Phi_{L_k}} \mathbb{Z}.$$

Proof. Let's write $L_i = S_i : D_i$ for each $1 \leq i \leq k$. The first claim holds because if
$$X \in [0, S_1 \star \cdots \star S_k)$$
corresponds to
$$X_1 \star \cdots \star X_k \in [0, S_1) \times \cdots \times [0, S_k)$$
under the canonical isomorphism $[0, S_1 \star \cdots \star S_k) \cong [0, S_1) \times \cdots \times [0, S_k)$, then
$$\begin{aligned}
\varphi_{L_1 \star \cdots \star L_k}(X) &= X \cdot (D_1 \star \cdots \star D_k) \\
&= (X_1 \star \cdots \star X_k) \cdot (D_1 \star \cdots \star D_k) \\
&= (X_1 \cdot D_1) + \cdots + (X_k \cdot D_k) \\
&= \varphi_{L_1}(X_1) + \cdots + \varphi_{L_k}(X_k).
\end{aligned}$$

For the second claim, we argue that the diagram

$$
\begin{array}{ccc}
[0, N_1) \times \cdots \times [0, N_1) & \xrightarrow{\mathrm{colex}^{-1}_{S_1} \times \cdots \times \mathrm{colex}^{-1}_{S_k}} & [0, S_1) \times \cdots \times [0, S_1) \\
{\scriptstyle \mathrm{colex}^{-1}_{(N_1, \ldots, N_k)}} \uparrow & \cong \uparrow & \searrow {\scriptstyle \varphi_{L_1} + \cdots + \varphi_{L_k}} \\
[0, N_1 \cdots N_k) \xrightarrow{\mathrm{colex}^{-1}_{S_1 \star \cdots \star S_k}} & [0, S_1 \star \cdots \star S_k) \xrightarrow{\varphi_{L_1 \star \cdots \star L_k}} & \mathbb{Z}
\end{array}
$$

commutes. The left-hand square commutes since colexicographic isomorphisms are associative, and the right-hand triangle commutes by the first claim. $\square$

We can describe the important attributes of a concatenated layout as follows.

Proposition 2.1.3.41. *Suppose $L_1, \ldots, L_k$ are flat layouts. Then*

1. the rank of $L_1 \star \cdots \star L_k$ is

$$\mathsf{rank}(L_1 \star \cdots \star L_k) = \sum_{i=1}^{k} \mathsf{rank}(L_i),$$

2. the size of $L_1 \star \cdots \star L_k$ is

$$\mathsf{size}(L_1 \star \cdots \star L_k) = \prod_{i=1}^{k} \mathsf{size}(L_i),$$

3. the cosize of $L_1 \star \cdots \star L_k$ is

$$\mathsf{cosize}(L_1 \star \cdots \star L_k) = 1 - k + \sum_{i=1}^{k} \mathsf{cosize}(L_i).$$

Proof. Let's write $L_i = S_i : D_i$ for each $1 \leq i \leq k$. For 1, we compute

$$\mathsf{rank}(L_1 \star \cdots \star L_k) = \mathsf{len}(S_1 \star \cdots \star S_k) = \sum_{i=1}^{k} \mathsf{len}(S_i) = \sum_{i=1}^{k} \mathsf{rank}(L_i).$$

For 2, we compute

$$\mathsf{size}(L_1 \star \cdots \star L_k) = \mathsf{size}(S_1 \star \cdots \star S_k) = \prod_{i=1}^{k} \mathsf{size}(S_i) = \prod_{i=1}^{k} \mathsf{size}(L_i).$$

For 3, we compute

$$\begin{aligned}
\mathsf{cosize}(L_1 \star \cdots \star L_k) &= 1 + \mathsf{max}(\Phi_{L_1 \star \cdots \star L_k}) \\
&= 1 + \sum_{i=1}^{k} \mathsf{max}(\Phi_{L_i}) \\
&= 1 - k + (1 + \mathsf{max}(\Phi_{L_1})) + \cdots + (1 + \mathsf{max}(\Phi_{L_1})) \\
&= 1 - k + \mathsf{cosize}(L_1) + \cdots + \mathsf{cosize}(L_k).
\end{aligned}$$

where we have used our identification of $\Phi_{L_1 \star \cdots \star L_k}$ from Proposition 2.1.3.40. $\qquad \square$

2.1.4 Flat coalesce

We have seen that the layout function Φ_L of a flat layout L is an important invariant. In many cases, we are only interested in the layout function Φ_L, and are free to work with any layout whose layout function is Φ_L. The flat coalesce operation

$$L \mapsto \mathsf{coal}^\flat(L)$$

provides us with the simplest flat layout whose layout function is Φ_L (see Proposition 2.1.4.19). We begin by defining the notion of a coalesced flat layout.

Definition 2.1.4.1. Suppose $L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$ is a flat layout. We say L is *coalesced* if

    1. for any $1 \leq i \leq m$, we have $s_i \neq 1$, and

    2. for any $1 \leq i < m$, we have $s_i d_i \neq d_{i+1}$.

Example 2.1.4.2. The flat layout

$$L = (3, 5, 2) : (7, 21, 4)$$

is not coalesced because $3 \cdot 7 = 21$.

Example 2.1.4.3. The flat layout

$$L = (2, 7, 6) : (1, 3, 10)$$

is coalesced.

Example 2.1.4.4. The empty layout $E = () : ()$ is coalesced.

Example 2.1.4.5. If $L = (s) : (d)$ and $s \neq 1$, then L is coalesced.

Example 2.1.4.6. If $L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$ is a column-major layout with $\mathsf{rank}(L) > 1$, then L is not coalesced, since for any $1 \leq i < m$, we have

$$s_i d_i = s_i(s_1 \cdots s_{i-1}) = s_1 \cdots s_i = d_{i+1}.$$

Example 2.1.4.7. If $L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$ is a row-major layout with $s_i > 1$ for all $1 \leq i \leq m$, then L is coalesced: If $1 \leq i < m$, then

$$s_i d_i = s_i s_{i+1} \cdots s_m > s_{i+2} \cdots s_m = d_{i+1}.$$

Example 2.1.4.8. A flat layout of the form

$$L = (s_1, \ldots, s_m) : (0, \ldots, 0)$$

is coalesced if and only if $m \leq 1$.

If L is a flat layout, then we may obtain a coalesced layout $\mathsf{coal}^\flat(L)$ with the same layout function as L by removing modes with $s_i = 1$, and combining modes with $s_i d_i = d_{i+1}$. More precisely, we make the following construction.

Construction 2.1.4.9. Suppose L is a flat layout, and write

$$\mathsf{squeeze}(L) = (s_1, \ldots, s_m) : (d_1, \ldots, d_m).$$

Let $\sim$ be the equivalence relation on $\langle m \rangle$ generated by $i \sim i+1$ if

$$s_i d_i = d_{i+1}.$$

The quotient $\langle m \rangle / \sim$ is ordered by $[i] \leq [i']$ if $i \leq i'$, so we may identify $\langle m \rangle / \sim$ with $\langle \bar{m} \rangle$, where $\bar{m}$ is the size of $\langle m \rangle / \sim$. If $i \in \langle \bar{m} \rangle$ corresponds to the equivalence class

$$I = \{i', i' + 1, \ldots, i' + k\} \in \langle m \rangle / \sim,$$

then we define integers $\bar{s}_i$ and $\bar{d}_i$ as

$$\bar{s}_i = s_{i'} s_{i'+1} \cdots s_{i'+k}$$

and

$$\bar{d}_i = d_{i'},$$

and define

$$\mathsf{coal}^\flat(L) = (\bar{s}_1, \ldots, \bar{s}_{\bar{m}}) : (\bar{d}_1, \ldots, \bar{d}_{\bar{m}}).$$

Observation 2.1.4.10. Examining the definition, we could equivalently define $\mathsf{coal}^\flat(L)$ to be the flat layout obtained from

$$\mathsf{squeeze}(L) = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

by iteratively performing the operation

$$s_i, s_{i+1} : d_i, s_i d_i \qquad \rightsquigarrow \qquad s_i s_{i+1} : d_i$$

until the result is coalesced.

Example 2.1.4.11. If $L = (2, 2, 2, 2, 2) : (8, 16, 1024, 2048, 4096)$, then

$$\mathsf{coal}^\flat(L) = (4, 8) : (8, 1024).$$

Example 2.1.4.12. If $L = (3, 4, 1, 5) : (1, 8, 3, 32)$, then

$$\mathsf{coal}^\flat(L) = (3, 20) : (1, 8).$$

Example 2.1.4.13. If $L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$ is column-major, and not all s_i are equal to 1, then

$$\mathsf{coal}^\flat(L) = (s_1 \cdots s_m) : (1).$$

Example 2.1.4.14. If L is row-major, then

$$\mathsf{coal}^\flat(L) = \mathsf{squeeze}(L).$$

Let's justify that the operation $L \mapsto \mathsf{coal}^\flat(L)$ results in a coalesced layout.

Lemma 2.1.4.15. *If L is a flat layout, then $\mathsf{coal}^\flat(L)$ is coalesced.*

Proof. Borrowing the notation of Construction 2.1.4.9, let

$$\mathsf{squeeze}(L) = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

and let

$$\mathsf{coal}^\flat(L) = (\bar{s}_1, \ldots, \bar{s}_{\bar{m}}) : (\bar{d}_1, \ldots, \bar{d}_{\bar{m}}).$$

We want to show that $\mathsf{coal}^\flat(L)$ is coalesced. Suppose $1 \le i \le \bar{m}$. Then i corresponds to a (non-empty) equivalence class $I \in \langle m \rangle / \sim$, and

$$\bar{s}_i = \prod_{i' \in I} s_{i'}$$

is a product of integers $s_{i'} > 1$, so $\bar{s}_i > 1$.

Suppose $1 \le i < \bar{m}$. We claim that $\bar{s}_i \bar{d}_i \ne \bar{d}_{i+1}$. Suppose i corresponds to the equivalence class

$$\{i', i'+1, \dots, i'+k\} \in \langle m \rangle / \sim,$$

and suppose $i+1$ corresponds to the equivalence class

$$\{i'+k+1, i'+k+2, \dots, i'+k+\ell\} \in \langle m \rangle / \sim .$$

Then by using the equalities $s_{i'+t} d_{i'+t} = d_{i'+t+1}$ for $0 \le t < k$, we may write

$$\begin{aligned}
\bar{s}_i \bar{d}_i = \bar{d}_i \bar{s}_i &= d_{i'} s_{i'} s_{i'+1} \cdots s_{i'+k} \\
&= d_{i'+1} s_{i'+1} \cdots s_{i'+k} \\
&\vdots \\
&= d_{i'+k} s_{i'+k} \\
&= s_{i'+k} d_{i'+k}
\end{aligned}$$

and since $i'+k$ and $i'+k+1$ do not lie in the same equivalence class, we have

$$\bar{s}_i \bar{d}_i = s_{i'+k} d_{i'+k} \ne d_{i'+k+1} = \bar{d}_{i+1}.$$

$\square$

Example 2.1.4.16. If L is coalesced, then $\mathsf{coal}^\flat(L) = L$. In particular, this implies that $\mathsf{coal}^\flat(-)$ is an idempotent operation:

$$\mathsf{coal}^\flat(\mathsf{coal}^\flat(L)) = \mathsf{coal}^\flat(L).$$

Next, we argue that coalescing a flat layout leaves the layout function unchanged.

Lemma 2.1.4.17. *If L is a flat layout, then $\Phi_{\mathsf{coal}^\flat(L)} = \Phi_L$.*

Proof. By Observation 2.1.4.10, it suffices to show that replacing an instance of $s_i, s_{i+1} : d_i, s_i d_i$ with $s_i s_{i+1} : d_i$ leaves the layout function unchanged. Suppose

$$L = (s_1, \dots, s_m) : (d_1, \dots, d_m)$$

is a flat layout, and there exists some $1 \le i < m$ such that $d_{i+1} = s_i d_i$. Let

$$L' = (s'_1, \dots, s'_{m-1}) : (d'_1, \dots, d'_{m-1})$$

denote the flat layout obtained from L by combining the ith and $(i+1)$th modes of L. More precisely, we have

$$s'_j = \begin{cases} s_j & j < i \\ s_i s_{i+1} & j = i \\ s_{j+1} & i < j < m, \end{cases} \qquad \text{and} \qquad d'_j = \begin{cases} d_j & j \le i \\ d_{j+1} & i < j < m. \end{cases}$$

The layout function for L is given by

$$\Phi_L(x) = x_1 d_1 + \cdots + x_m d_m$$

where $x_j = \left\lfloor \dfrac{x}{s_1 \cdots s_{j-1}} \right\rfloor \mod s_j$, and the layout function for L' is given by

$$\Phi_{L'}(x) = x'_1 d'_1 + \cdots + x'_{m-1} d'_{m-1}$$

where $x'_j = \left\lfloor \dfrac{x}{s'_1 \cdots s'_{j-1}} \right\rfloor \mod s'_j$. We observe that

$$x'_j = \begin{cases} x_j & j < i \\ x_i + x_{i+1}s_i & j = i \\ x_{j+1} & i < j < m, \end{cases}$$

and so

$$\begin{aligned} \Phi_L(x) &= x_1 d_1 + \cdots + x_m d_m \\ &= x_1 d_1 + \cdots + x_i d_i + x_{i+1} s_i d_i + \cdots + x_m d_m \\ &= x_1 d_1 + \cdots + (x_i + x_{i+1} s_i) d_i + \cdots + x_m d_m \\ &= x'_1 d'_1 + \cdots + x'_{m-1} d'_{m-1} \\ &= \Phi_{L'}(x). \end{aligned}$$

$\square$

We can use the coalesce operation to characterize when two flat layouts have the same layout function.

Proposition 2.1.4.18. *Suppose A and B are flat layouts. Then*

$$\Phi_A = \Phi_B \quad \Leftrightarrow \quad \mathsf{coal}^\flat(A) = \mathsf{coal}^\flat(B).$$

Proof. If $\mathsf{coal}^\flat(A) = \mathsf{coal}^\flat(B)$, then by Lemma 2.1.4.17, we have

$$\Phi_A = \Phi_{\mathsf{coal}^\flat(A)} = \Phi_{\mathsf{coal}^\flat(B)} = \Phi_B.$$

Inversely, suppose that $\mathsf{coal}^\flat(A) \neq \mathsf{coal}^\flat(B)$. We will argue that $\Phi_A \neq \Phi_B$. Let's write

$$\mathsf{coal}^\flat(A) = (s_1, \ldots, s_m) : (d_1, \ldots, d_m),$$
$$\mathsf{coal}^\flat(B) = (t_1, \ldots, t_n) : (e_1, \ldots, e_m).$$

If one of m, n is nonzero and the other is 0, then clearly $\Phi_A \neq \Phi_B$, so we may assume $m, n \geq 1$. Let i denote the least integer such that $(s_i, d_i) \neq (t_i, e_i)$. Then, in particular, we have $s_1 \cdots s_j = t_1 \cdots t_j$ for any $j < i$. There are two cases to consider:

- (Case 1): Suppose $d_i \neq e_i$. Let $N = s_1 \cdots s_{i-1} = t_1 \cdots t_{i-1}$. Then

$$\Phi_{\mathsf{coal}^\flat(A)}(N) = d_i \neq e_i = \Phi_{\mathsf{coal}^\flat(B)}(N)$$

  so $\Phi_{\mathsf{coal}^\flat(A)} \neq \Phi_{\mathsf{coal}^\flat(B)}$, and hence $\Phi_A \neq \Phi_B$.

- (Case 2): Suppose $d_i = e_i$, so that $s_i \neq t_i$. Without loss of generality we may assume $s_i < t_i$. Let $N = s_1 \cdots s_i = (t_1 \cdots t_{i-1})s_i$. Then

$$\Phi_{\mathsf{coal}^\flat(A)}(N) = d_{i+1}$$

  while

$$\begin{aligned} \Phi_{\mathsf{coal}^\flat(B)}(N) &= s_i e_i \\ &= s_i d_i, \end{aligned}$$

  and since $\mathsf{coal}^\flat(A)$ is coalesced, we have $d_{i+1} \neq s_i d_i$. We deduce that $\Phi_{\mathsf{coal}^\flat(A)} \neq \Phi_{\mathsf{coal}^\flat(B)}$, and hence $\Phi_A \neq \Phi_B$.

$$\square$$

The previous proposition affords us the following abstract characterization of $\mathsf{coal}^\flat(L)$.

Proposition 2.1.4.19. *If L is a flat layout, then $\mathsf{coal}^\flat(L)$ is the unique flat layout of minimal rank whose layout function is Φ_L.*

Proof. Suppose L' is a flat layout with $\Phi_{L'} = \Phi_L$. Then by Proposition 2.1.4.18, we have

$$\mathsf{coal}^\flat(L) = \mathsf{coal}^\flat(L'),$$

and it follows that

$$\mathsf{rank}(\mathsf{coal}^\flat(L)) = \mathsf{rank}(\mathsf{coal}^\flat(L')) \le \mathsf{rank}(L'),$$

where equality holds if and only if

$$L' = \mathsf{coal}^\flat(L') = \mathsf{coal}^\flat(L).$$

$$\square$$

2.1.5 Compact flat layouts

Before treating layout complements, we must define an important family of layouts called *compact* flat layouts. These are the flat layouts whose layout functions are bijective. In terms of the standard grid diagrams depicting layouts, a flat layout L is compact if each integer $0 \le i < \mathsf{size}(L)$ appears exactly once. For instance, the layout

$$A = (3,6) : (1,3) =$$

0	3	6	9	12	15
1	4	7	10	13	16
2	5	8	11	14	17

is compact, while the layouts

$$B = (3,6) : (2,6) =$$

0	6	12	18	24	30
2	8	14	20	26	32
4	10	16	22	28	34

and

$$C = (3,6) : (1,2) =$$

0	2	4	6	8	10
1	3	5	7	9	11
2	4	6	8	10	12

are not compact. More precisely, we have the following definition.

Definition 2.1.5.1. Suppose L is a flat layout. We say L is *compact* if

$$[0, \mathsf{size}(L)) \xrightarrow{\Phi_L^{\mathsf{cosize}(L)}} [0, \mathsf{cosize}(L))$$

is an isomorphism.

Example 2.1.5.2. The flat layout

$$L = (2, 2, 2, 2) : (1, 2, 4, 8)$$

is compact. More generally, if L is column-major, then L is compact.

Example 2.1.5.3. The flat layout

$$L = (3, 64, 32) : (2048, 32, 1)$$

is compact. More generally, if L is row-major, then L is compact.

Example 2.1.5.4. The empty layout
$$E = (\,) : (\,)$$
is compact.

Example 2.1.5.5. Suppose

$$L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

is a flat layout. If there is some mode of L with $s_i > 1$ and $d_i = 0$, then L is not compact.

We can give an explicit characterization of compact layouts as follows.

Proposition 2.1.5.6. *Suppose L is a flat layout, and write*

$$\mathsf{squeeze}(L) = (s_1, \ldots, s_m) : (d_1, \ldots, d_m).$$

then L is compact if and only if there exists a permutation $\sigma \in \Sigma_m$ such that

$$d_{\sigma(i)} = s_{\sigma(1)} \cdots s_{\sigma(i-1)}$$

for all $1 \le i \le m$. In other words, L is compact if and only if there exists a permutation $\sigma \in \Sigma_m$ such that $\mathsf{squeeze}(L)^\sigma$ is column-major.

Proof. Suppose L is a flat layout, and write

$$\mathsf{squeeze}(L) = (s_1, \ldots, s_m) : (d_1, \ldots, d_m).$$

Suppose first that L is compact, so there exists a permutation $\sigma \in \Sigma_m$ such that $d_{\sigma(i)} = s_{\sigma(1)} \cdots s_{\sigma(i-1)}$ for each $1 \le i \le m$. If we write $S^\sigma = (s_{\sigma(1)}, \ldots, s_{\sigma(m)})$, then we can write $\Phi_L^{\mathsf{cosize}(L)}$ as the composite

$$[0, \mathsf{size}(L)) \xrightarrow{\mathsf{colex}_S^{-1}} [0, S) \xrightarrow{\cong} [0, S^\sigma) \xrightarrow{\mathsf{colex}_{S^\sigma}} [0, \mathsf{cosize}(L))$$

$$(x_1, \ldots, x_m) \longmapsto (x_{\sigma(1)}, \ldots, x_{\sigma(m)})$$

and since each of these maps is an isomorphism, so is the composite $\Phi_L^{\mathsf{cosize}(L)}$.

Conversely, suppose that $\Phi_L^{\mathsf{cosize}(L)}$ is an isomorphism. First, we note that the strides $d_1, \ldots, d_m$ must be pairwise distinct: Suppose $d_i = d_j$, and let δ_i^m and δ_j^m denote the tuples whose ith (resp. jth) entry is 1, and all of whose other entries are 0. These tuples satisfy

$$\delta_i^m \cdot (d_1, \ldots, d_m) = d_i = d_j = \delta_j^m \cdot (d_1, \ldots, d_m),$$

and since $\Phi_L^{\mathsf{cosize}(L)}$ is injective, we must have $i = j$. Given that the strides $d_1, \ldots, d_m$ are pairwise distinct, let $\sigma \in \Sigma_m$ be the permutation such that

$$d_{\sigma(1)} < d_{\sigma(2)} < \cdots < d_{\sigma(m)}.$$

We will argue by induction on $i \geq 1$ that $d_{\sigma(i)} = s_{\sigma(1)} \cdots s_{\sigma(i-1)}$. For the base case $i = 1$, we note that 1 is in the image of $\Phi_L^{\mathsf{cosize}(L)}$, and the smallest non-zero value of $\Phi_L^{\mathsf{cosize}(L)}$ is $d_{\sigma(1)}$, so it follows that $d_{\sigma(1)} = 1$. Suppose $i > 1$, and that we have proved the claim for all $j < i$. Consider the stride $d_{\sigma(i)}$. We know that there is no tuple of the form $(x_1, \ldots, x_{i-1}, 0, \ldots, 0)^\sigma$ such that

$$(x_1, \ldots, x_{i-1}, 0, \ldots, 0)^\sigma \cdot (d_1, \ldots, d_m) = s_{\sigma(1)} \cdots s_{\sigma(i-1)},$$

since the largest possible value of such an expression is

$$\sum_{j=1}^{i-1} (s_{\sigma(j)} - 1)(s_{\sigma(1)} \cdots s_{\sigma(j-1)}) = s_{\sigma(1)} \cdots s_{\sigma(i-1)} - 1.$$

Since $\Phi_L^{\mathsf{cosize}}(L)$ is surjective, and $d_{\sigma(i)} < d_{\sigma(i+1)} < \cdots < d_{\sigma(m)}$, it follows that the next largest value of $\Phi_L^{\mathsf{cosize}(L)}$ is $d_{\sigma(i)}$, so we must have $d_{\sigma(i)} = s_{\sigma(1)} \cdots s_{\sigma(i-1)}$, as claimed. $\qquad \square$

We conclude this section by giving a family of equivalent conditions for a flat layout L to be compact.

Proposition 2.1.5.7. *Suppose L is a flat layout. Then the following are equivalent.*

1. *L is compact.*

2. *$\mathsf{coal}^\flat(L)$ is compact.*

3. *$\mathsf{squeeze}(L)$ is compact.*

4. *$\mathsf{sort}(L)$ is compact.*

Proof. The equivalence of 1, 2, and 3, follows from the fact that

$$\Phi_L = \Phi_{\mathsf{coal}^\flat(L)} = \Phi_{\mathsf{squeeze}(L)}.$$

It remains to prove that L is compact if and only if $\mathsf{sort}(L)$ is compact. Using the fact that

$$\mathsf{squeeze}(\mathsf{sort}(L)) = \mathsf{sort}(\mathsf{squeeze}(L)),$$

we have

$$
\begin{aligned}
\mathsf{sort}(L) \text{ is compact.} \quad &\Leftrightarrow \quad \mathsf{squeeze}(\mathsf{sort}(L)) \text{ is compact.} \\
&\Leftrightarrow \quad \mathsf{sort}(\mathsf{squeeze}(L)) \text{ is compact.}
\end{aligned}
$$

Now $\mathsf{sort}(\mathsf{squeeze}(L)) = \mathsf{squeeze}(L)^\tau$ for some permutation $\tau \in \Sigma_m$, so there exists a permutation σ such that $\mathsf{squeeze}(L)^\sigma$ is column-major if and only if there exists a permutation $\sigma' \in \Sigma_m$ such that $\mathsf{sort}(\mathsf{squeeze}(L))$ is column-major, namely $\sigma' = \tau^{-1}\sigma$. It follows that

$$
\begin{aligned}
\mathsf{sort}(\mathsf{squeeze}(L)) \text{ is compact.} &\Leftrightarrow \quad \mathsf{squeeze}(L) \text{ is compact.} \\
&\Leftrightarrow \quad L \text{ is compact.}
\end{aligned}
$$

$\qquad \square$

2.1.6 Complements

In this section, we define the notion of complementary flat layouts. Recall from Definition 2.1.5.1 that a flat layout L is *compact* if the layout function

$$\Phi_L^{\mathsf{cosize}(L)} : [0, \mathsf{size}(L)) \to [0, \mathsf{cosize}(L))$$

is an isomorphism.

Definition 2.1.6.1. Suppose A and B are flat layouts. We say B is a *complement* of A, and write $A \perp B$, if the concatenated layout $A \star B$ is compact.

Example 2.1.6.2. If $A = (3) : (5)$ and $B = (5) : (1)$, then $A \perp B$ since

$$A \star B = (3, 5) : (5, 1)$$

is compact.

$$
B = \quad
\begin{array}{|c|c|c|c|c|}
\hline
0 & 1 & 2 & 3 & 4 \\
\hline
\end{array}
$$

$$
A = \quad
\begin{array}{|c|}
\hline
0 \\
\hline
5 \\
\hline
10 \\
\hline
\end{array}
\qquad
\begin{array}{|c|c|c|c|c|}
\hline
0 & 1 & 2 & 3 & 4 \\
\hline
5 & 6 & 7 & 8 & 9 \\
\hline
10 & 11 & 12 & 13 & 14 \\
\hline
\end{array}
\quad = A \star B
$$

Example 2.1.6.3. If $A = (4, 2, 10) : (1400, 2, 20)$ and $B = (2, 5, 7, 2) : (1, 4, 200, 5600)$, then $A \perp B$ since

$$A \star B = (4, 2, 10, 2, 5, 7, 2) : (1400, 2, 20, 1, 4, 200, 5600)$$

is compact.

Example 2.1.6.4. If A is a flat layout and $E = () : ()$ is the empty layout, then $A \perp A$ if and only if A is compact, since

$$A \star E = A.$$

Example 2.1.6.5. If A and B are flat layouts, then

$$A \perp B \quad \Leftrightarrow \quad B \perp A.$$

Example 2.1.6.6. If A is a flat layout, then $A \perp A$ if and only if $\mathsf{size}(A) = 1$.

Observation 2.1.6.7. In order for A to admit a complement, it is necessary that Φ_A is injective. There do, however, exist flat layouts A such that Φ_A is injective, and A does not admit a complement. For example, consider the layout

$$A = (2, 2) : (1, 3).$$

The layout function of A is injective since

$$\Phi_A(0) = 0, \ \Phi_A(1) = 1, \ \Phi_A(2) = 3, \text{ and } \Phi_A(3) = 4,$$

but A does not admit a complement: Suppose

$$B = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

is any other flat layout. If there does not exist a tuple

$$(x_1, x_2, y_1, \ldots, y_m) \in [0, 2) \times [0, 2) \times [0, s_1) \times \cdots \times \cdots [0, s_m)$$

such that $\varphi_{A \star B}(x_1, x_2, y_1, \ldots, y_m) = 2$, then $A \star B$ is not compact. Suppose otherwise that there is such a tuple $(x_1, x_2, y_1, \ldots, y_m)$. Then $\varphi_B(y_1, \ldots, y_m) \in \{0, 1, 2\}$.

- (Case 1): If $\varphi_B(y_1, \ldots, y_m) = 0$, then

$$\varphi_{A \star B}(0, 0, 0, \ldots, 0) = 0 = \varphi_{A \star B}(0, 0, y_1, \ldots, y_m).$$

- (Case 2): If $\varphi_B(y_1, \ldots, y_m) = 1$, then

$$\varphi_{A \star B}(1, 0, 0, \ldots, 0) = 1 = \varphi_{A \star B}(0, 0, y_1, \ldots, y_m).$$

- (Case 3): If $\varphi_B(y_1, \ldots, y_m) = 2$, then

$$\varphi_{A \star B}(0, 1, 0, \ldots, 0) = 3 = \varphi_{A \star B}(1, 0, y_1, \ldots, y_m).$$

In any case, we deduce that $\varphi_{A \star B}$ is not injective, hence neither is $\Phi_{A \star B}$. This implies that $A \star B$ is not compact, so B is not a complement of A.

Observation 2.1.6.8. Complements are not unique. For example, if

$$A = (8, 8) : (2, 32),$$

then each of the layouts

$$B_1 = (2, 2) : (1, 16)$$
$$B_2 = (2, 2) : (16, 1)$$
$$B_3 = (5, 2, 2, 1) : (256, 1, 16, 0)$$

is a complement of A. Instead, there is a (possibly empty) set

$$\mathsf{complements}^\flat(A) = \{\text{flat layouts } B \mid B \text{ is a complement of } A\}.$$

of layouts which are complementary to A.

It will be useful to provide a family of equivalent conditions for B to be a complement of A (see Proposition 2.1.6.10). In order to do so, we need the following technical lemma, which describes the interplay between concatenation, and the operations $\mathsf{squeeze}(-)$, $\mathsf{sort}(-)$, and $\mathsf{coal}^\flat(-)$.

Lemma 2.1.6.9. *Suppose A and B are flat layouts. Then*

1. $\mathsf{squeeze}(A \star B) = \mathsf{squeeze}(A) \star \mathsf{squeeze}(B)$,

2. $\mathsf{sort}(A \star B) = \mathsf{sort}(L \star \mathsf{sort}(B))$, *and*

3. $\mathsf{coal}^\flat(A \star B) = \mathsf{coal}^\flat(A \star \mathsf{coal}^\flat(B))$.

Proof. Write

$$A = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$
$$B = (t_1, \ldots, t_n) : (e_1, \ldots, e_n).$$

If we let $\{i_1 < \cdots < i_{m'}\} \subset \langle m \rangle$ denote the indices with $s_{i_k} \neq 1$, and $\{j_1, \ldots, j_{n'}\} \subset \langle n \rangle$ denote the indices with $t_{j_\ell} \neq 1$, then

$$\mathsf{squeeze}(A \star B) = (s_{i_1}, \ldots, s_{i_{m'}}, t_{j_1}, \ldots, t_{j_{n'}}) : (d_{i_1}, \ldots, d_{i_{m'}}, e_{j_1}, \ldots, e_{j_{n'}})$$
$$= \mathsf{squeeze}(A) \star \mathsf{squeeze}(B).$$

This proves 1. For 2, we note that for any flat layout L, and any permutation $\sigma \in \Sigma_{\mathsf{len}(L)}$, we have $\mathsf{sort}(L) = \mathsf{sort}(L^\sigma)$. The result follows from the observation that

$$A \star \mathsf{sort}(B) = (A \star B)^\sigma$$

where σ is a block permutation of the form $\sigma = \mathsf{id} \times \sigma' \in \Sigma_m \times \Sigma_n \subset \Sigma_{m+n}$. For 3., it suffices to show that $A \star B$ and $A \star \mathsf{coal}^\flat(B)$ have the same layout function. This follows from Proposition 2.1.3.40. $\quad\square$

Proposition 2.1.6.10. *Suppose A and B are flat layouts. Then the following are equivalent.*

1. *$A \perp B$.*

2. *$B \perp A$.*

3. *$A \perp \mathsf{squeeze}(B)$.*

4. *$A \perp \mathsf{coal}^\flat(B)$.*

5. *$A \perp \mathsf{sort}(B)$.*

Proof. We use Proposition 2.1.5.7 and Lemma 2.1.6.9 to prove the equivalence of these conditions. First, we note that $\mathsf{sort}(A \star B) = \mathsf{sort}(B \star A)$, which implies the equivalence of 1 and 2. Next, we note that, by Lemma 2.1.6.9, if $\mathsf{op}(-)$ is any of the operations $\mathsf{squeeze}(-)$, $\mathsf{sort}(-)$, or $\mathsf{coal}^\flat(-)$, then

$$\mathsf{op}(A \star B) = \mathsf{op}(A \star \mathsf{op}(B)),$$

and so

$$
\begin{aligned}
A \perp B \quad &\Leftrightarrow \quad A \star B \text{ is compact.} \\
&\Leftrightarrow \quad \mathsf{op}(A \star B) \text{ is compact.} \\
&\Leftrightarrow \quad \mathsf{op}(A \star \mathsf{op}(B)) \text{ is compact.} \\
&\Leftrightarrow \quad \mathsf{op}(B) \text{ is a complement of } A.
\end{aligned}
$$

$\quad\square$

We would like to characterize when a flat layout admits a complement. To this end, we make the following definition.

Definition 2.1.6.11. Suppose A is a flat layout, and write

$$\mathsf{sort}(\mathsf{squeeze}(A)) = (s_1, \ldots, s_m) : (d_1, \ldots, d_m).$$

We say A is *complementable* if for any $1 \leq i < m$, the integer $s_i d_i$ divides d_{i+1}.

Example 2.1.6.12. The flat layout

$$A_1 = (4, 1, 1, 4, 4) : (64, 0, 0, 1, 8)$$

is complementable, while the flat layout

$$A_2 = (4, 4, 4) : (64, 1, 1)$$

is not complementable.

Example 2.1.6.13. The flat layout

$$A_1 = (10, 2) : (4, 80)$$

complementable, while the flat layout

$$A_2 = (10, 2) : (80, 4)$$

is not complementable.

Example 2.1.6.14. If A is compact, then by Proposition 2.1.5.6, A is complementable.

Example 2.1.6.15. Suppose

$$A = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

is a flat layout. If there is any $1 \leq i \leq m$ such that $s_i \neq 1$ and $d_i = 0$, then A is not complementable.

If A is complementable, then we can construct a complement of A as follows.

Construction 2.1.6.16. Suppose A is a flat layout, and write

$$\mathsf{sort}(\mathsf{squeeze}(A)) = (s_1, \ldots, s_m) : (d_1, \ldots, d_m).$$

If A is complementable, then we define a flat layout $\mathsf{comp}^\flat(A)$ as

$$\mathsf{comp}^\flat(A) = \mathsf{coal}^\flat(C)$$

where

$$C = \left(d_1, \frac{d_2}{s_1 d_1}, \frac{d_3}{s_2 d_2}, \ldots, \frac{d_m}{s_{m-1} d_{m-1}} \right) : \left(1, s_1 d_1, s_2 d_2, \ldots, s_{m-1} d_{m-1} \right).$$

Example 2.1.6.17. If $A = (8, 8) : (1, 8)$, then

$$\mathsf{comp}^\flat(A) = () : ()$$

is the empty layout. More generally, if A is compact, then $\mathsf{comp}^\flat(A) = () : ()$ is the empty layout.

Example 2.1.6.18. If $A = (2, 2) : (2, 8)$, then

$$\mathsf{comp}^\flat(A) = (2, 2) : (1, 4).$$

Example 2.1.6.19. If $A = (3, 3, 8) : (16, 96, 1)$, then

$$\mathsf{comp}^\flat(A) = (2, 2) : (8, 48).$$

Let's justify that $\mathsf{comp}^\flat(A)$ is, in fact, a complement of A.

Lemma 2.1.6.20. *Suppose A is a flat layout. If A is a complementable, then*

$$A \perp \mathsf{comp}^\flat(A).$$

Proof. Lets write

$$\mathsf{sort}(\mathsf{squeeze}(A)) = (s_1, \ldots, s_m) : (d_1, \ldots, d_m),$$

so that $\mathsf{comp}^\flat(A) = \mathsf{coal}^\flat(C)$ where

$$C = \left(d_1, \frac{d_2}{s_1 d_1}, \frac{d_3}{s_2 d_2}, \ldots, \frac{d_m}{s_{m-1} d_{m-1}}\right) : \left(1, s_1 d_1, s_2 d_2, \ldots, s_{m-1} d_{m-1}\right).$$

By Proposition 2.1.6.10, it suffices to prove that C is a complement of $\mathsf{sort}(\mathsf{squeeze}(A))$. This is the case since the concatenation

$$\mathsf{sort}(\mathsf{squeeze}(A)) \star C$$

is equal to

$$\left(s_1, \ldots, s_m, d_1, \frac{d_2}{s_1 d_1}, \ldots, \frac{d_m}{s_{m-1} d_{m-1}}\right) : (d_1, \ldots, d_m, 1, s_1 d_1, \ldots, s_{m-1} d_{m-1}),$$

and its sorting is equal to

$$\left(d_1, s_1, \frac{d_2}{s_1 d_1}, \ldots, \frac{d_m}{s_{m-1} d_{m-1}}, s_m\right) : (1, d_1, s_1 d_1, \ldots, s_{m-1} d_{m-1}, d_m)$$

which is column-major. $\qquad\square$

We have shown that if A is complementable, then A admits a complement. Next, we prove that the converse also holds.

Proposition 2.1.6.21. *Suppose A is a flat layout. Then there exists a complement B of A if and only if A is complementable.*

Proof. If A is complementable, then by Lemma 2.1.6.20 the layout $B = \mathsf{comp}^\flat(A)$ is a complement of A. Conversely, suppose there exists a complement B of A, and consider the flat layout

$$L = \mathsf{sort}\big(\mathsf{squeeze}(A) \star \mathsf{squeeze}(B)\big)$$
$$= (s_1, \ldots, s_m) : (d_1, \ldots, d_n).$$

Since $\Phi_L(0) = 0$, and Φ_L is injective, we know that $d_1 \neq 0$. We will argue that $d_i = s_1 \cdots s_{i-1}$, i.e., that L is column-major. Since

$$\Phi_L^{\mathsf{cosize}(L)} : [0, \mathsf{size}(L)) \to [0, \mathsf{cosize}(L))$$

is a bijection, we know that 1 is in the image of Φ_L, which implies that $d_1 = 1$. Suppose $1 < i \leq m$, and suppose we have proved that $d_j = s_1 \cdots s_{j-1}$ for all $j < i$. Consider the stride d_i. We know that there is no $(x_1, \ldots, x_{i-1}, 0, \ldots, 0)$ such that $(x_1, \ldots, x_{i-1}, 0, \ldots, 0) \cdot (d_1, \ldots, d_m) = s_1 \cdots s_{i-1}$, since the largest possible value of such an expression is

$$\sum_{j=1}^{i-1} (s_j - 1)(s_1 \cdots s_{j-1}) = s_1 \cdots s_i - 1.$$

Since Φ_L is surjective, and $d_i \leq d_{i+1} \leq \cdots \leq d_m$, it follows that the next largest value of Φ_L is d_i, so we must have $d_i = s_1 \cdots s_{i-1}$, as claimed.

Returning to our main goal, consider the layout

$$\mathsf{sort}(\mathsf{squeeze}(A)) = (s'_1, \ldots, s'_{m'}) : (d'_1, \ldots, d'_{m'}).$$

Then there exist $j_1 < \cdots < j_{m'}$ such that $s'_i = s_{j_i}$ and $d'_i = d_{j_i}$ for each $1 \leq i \leq m'$. If $1 \leq i < m'$, then

$$s'_i d'_i = s_{j_i} d_{j_i} = s_{j_i} s_1 \cdots s_{j_i - 1}$$

divides

$$d'_{i+1} = s_1 \cdots s_{j_{i+1} - 1},$$

so we conclude that A is complementable. $\qquad\square$

Our next goal is to give an abstract characterization of the complement $\mathsf{comp}^\flat(A)$ of a flat layout A. In order to do so, we need the following lemma.

Lemma 2.1.6.22. *Suppose A is a flat layout. If A is complementable and sorted, then the layout function*

$$\Phi_A : [0, \mathsf{size}(A)) \to \mathbb{Z}$$

is increasing.

Proof. Write

$$A = (s_1, \ldots, s_m) : (d_1, \ldots, d_m).$$

If $1 \leq k \leq m$, we claim that

$$d_1(s_1 - 1) + d_2(s_2 - 1) + \cdots + d_{k-1}(s_{k-1} - 1) \leq d_k.$$

If $k = 1$, this holds vacuously, and by induction on k, we have

$$d_1(s_1 - 1) + \cdots + d_{k-2}(s_{k-2} - 1) + d_{k-1}(s_{k-1} - 1) \leq d_{k-1} + d_{k-1}(s_k - 1)$$
$$= d_{k-1} s_{k-1}$$
$$\leq d_k.$$

Now, suppose we have $x, y \in [0, \mathsf{size}(A))$ with $x \leq y$. These integers correspond, under the colexicographic isomorphism, to tuples.

$$(x_1, \ldots, x_m), (y_1, \ldots, y_m) \in [0, s_1) \times \cdots \times [0, s_m)$$

Since $x \leq y$, we know there is some maximal $1 \leq k \leq m$ such that $x_k < y_k$, and $x_\ell = y_\ell$ for all $k < \ell \leq m$. Now we can compute

$$\Phi_A(x) = d_1 x_1 + \cdots + d_{k-1} x_{k-1} + d_k x_k + d_{k+1} x_{k+1} + \cdots + d_m x_m$$
$$= d_1 x_1 + \cdots + d_{k-1} x_{k-1} + d_k x_k + d_{k+1} y_{k+1} + \cdots + d_m y_m$$
$$\leq d_1(s_1 - 1) + \cdots + d_{k-1}(s_{k-1} - 1) + d_k x_k + d_{k+1} y_{k+1} + \cdots + d_m y_m$$
$$\leq d_k + d_k x_k + d_{k+1} y_{k+1} + \cdots + d_m y_m$$
$$= d_k(x_k + 1) + d_{k+1} y_{k+1} + \cdots + d_m y_m$$
$$\leq d_k y_k + d_{k+1} y_{k+1} + \cdots + d_m y_m$$
$$\leq d_1 y_1 + \ldots d_m y_m$$
$$= \Phi_A(y).$$

$\qquad\square$

Proposition 2.1.6.23. *Suppose A and B are flat layouts. If*

1. *$A \perp B$,*

2. *$\mathsf{size}(B) = \mathsf{size}(\mathsf{comp}^\flat(A))$,*

3. *B is coalesced, and*

4. *B is sorted,*

then $B = \mathsf{comp}^\flat(A)$.

Proof. Conditions 1 and 2 imply that Φ_B and $\Phi_{\mathsf{comp}^\flat(A)}$ have the same image. Since B and $\mathsf{comp}^\flat(A)$ are sorted, we know by Lemma 2.1.6.22 that Φ_B and $\Phi_{\mathsf{comp}^\flat(A)}$ are increasing. Combining these two facts, it follows that $\Phi_B = \Phi_{\mathsf{comp}^\flat(A)}$. Proposition 2.1.4.18 and condition 3 then imply that

$$B = \mathsf{coal}^\flat(B) = \mathsf{coal}^\flat(\mathsf{comp}^\flat(A)) = \mathsf{comp}^\flat(A).$$

$\square$

Definition 2.1.6.24. Suppose A and B are flat layouts, and N is a positive integer. We say B is a *N-complement* of A if B is a complement of A and

$$\mathsf{size}(A) \cdot \mathsf{size}(B) = N.$$

Definition 2.1.6.25. Suppose A is a flat layout, and write

$$\mathsf{sort}(\mathsf{squeeze}(A)) = (s_1, \ldots, s_m) : (d_1, \ldots, d_m).$$

We say A is N-complementable if

1. for all $1 \le i < m$, the integer $s_i d_i$ divides d_{i+1}, and

2. the integer $s_m d_m$ divides N.

Observation 2.1.6.26. If A is complementable, and $s_m : d_m$ is the last mode in the layout $\mathsf{sort}(\mathsf{squeeze}(A))$, then A is N-complementable exactly when N is a positive integer multiple of $s_m d_m$.

Observation 2.1.6.27. N-complements are not unique. For example, if $A = (2,2) : (1,50)$ and $N = 100$, then each of the layouts $B_1 = (25) : (2)$, and $B_2 = (5,5) : (2,10)$ is a N-complement of A. As a more general example, if B is a N-complement of A, then $\mathsf{coal}^\flat(B)$ is also a N-complement of A.

Remark 2.1.6.28. Suppose A is a flat layout and B_1 and B_2 are N-complements of A. Then the layout functions Φ_{B_1} and Φ_{B_2} need not be equal, but they necessarily have the same image. For example, if $A = (4) : (63)$ and $N = 252$ then $B_1 = (7,9) : (1,7)$ and $B_2 = (9,7) : (7,1)$ are N-complements of A, and $\Phi_{B_1} \ne \Phi_{B_2}$, since

$$\Phi_{B_1}(1) = 1 \ne 7 = \Phi_{B_2}(1).$$

As a more general example, if B is a N-complement of A, then $\mathsf{sort}(B)$ is also a N-complement of A.

Construction 2.1.6.29. Suppose A is a flat layout, N is a positive integer, and A is N-complementable. If we write

$$\mathsf{sort}(\mathsf{squeeze}(A)) = (s_1, \ldots, s_m) : (d_1, \ldots, d_m),$$

then we define a flat layout $\mathsf{comp}^\flat(A, N)$ by

$$\mathsf{comp}^\flat(A, N) = \mathsf{coal}^\flat(C)$$

where

$$C = \left(d_1, \frac{d_2}{s_1 d_1}, \frac{d_3}{s_2 d_2}, \dots, \frac{N}{s_m d_m}\right) : \left(1, s_1 d_1, s_2 d_2, \dots, s_m d_m\right).$$

Example 2.1.6.30. If $A = (3, 10) : (80, 4)$ and $N = 2400$, then

$$\mathsf{comp}^\flat(A, N) = (4, 2, 10) : (1, 40, 240).$$

Lemma 2.1.6.31. *Suppose A is a flat layout, N is a positive integer, and A is N-complementable. Then $\mathsf{comp}^\flat(A, N)$ is a N-complement of A.*

Proof. Lets write

$$\mathsf{sort}(\mathsf{squeeze}(A)) = (s_1, \dots, s_m) : (d_1, \dots, d_m),$$

so that $\mathsf{comp}^\flat(A, N) = \mathsf{coal}^\flat(C)$ where

$$C = \left(d_1, \frac{d_2}{s_1 d_1}, \frac{d_3}{s_2 d_2}, \dots, \frac{N}{s_m d_m}\right) : (1, s_1 d_1, s_2 d_2, \dots, s_m d_m).$$

First, we compute

$$
\begin{aligned}
\mathsf{size}(A) \cdot \mathsf{size}(B) &= \left(\prod_{i=1}^{m} s_i\right) \cdot \left(d_1 \cdot \left(\prod_{i=2}^{m} \frac{d_i}{s_{i-1} d_{i-1}}\right) \cdot \frac{N}{s_m d_m}\right) \\
&= \frac{\left(\prod_{i=1}^{m} s_i\right)\left(\prod_{i=1}^{m} d_i\right)}{\left(\prod_{i=1}^{m} s_i d_i\right)} \cdot N \\
&= N.
\end{aligned}
$$

We need to check that $A \star B$ is compact. Equivalently, we need to check that $\Phi_{A \star B}^N$ is an isomorphism. By Lemma 2.1.5.6, it suffices to prove that

$$\mathsf{squeeze}(A) \star \mathsf{squeeze}(B)$$

is compact. This is the case since this layout is equal to

$$\left(s_1, \dots, s_m, d_1, \frac{d_2}{s_1 d_1}, \dots, \frac{N}{s_m d_m}\right) : (d_1, \dots, d_m, 1, s_1 d_1, \dots, s_m d_m)$$

and so its sorting

$$\mathsf{sort}(\mathsf{squeeze}(A) \star \mathsf{squeeze}(B))$$

is equal to

$$\left(d_1, s_1, \frac{d_2}{s_1 d_1}, \dots, \frac{d_m}{s_{m-1} d_{m-1}}, s_m, \frac{N}{s_m d_m}\right) : (1, d_1, s_1 d_1, \dots, s_{m-1} d_{m-1}, d_m, s_m d_m)$$

which is column-major. $\qquad\square$

Proposition 2.1.6.32. *Suppose A is a flat layout and N is a positive integer. Then there exists a N-complement B of A if and only if A is N-complementable.*

Proof. If A is N-complementable, then by Lemma 2.1.6.31 the layout $B = \mathsf{comp}^\flat(L, N)$ is a N-complement of A.

On the other hand, suppose there exists a N-complement B of A. Consider the flat layout

$$L := \mathsf{sort}\big(\mathsf{squeeze}(A) \star \mathsf{squeeze}(B)\big)$$
$$= (s_1, \ldots, s_m) : (d_1, \ldots, d_n).$$

Since $\Phi_L(0) = 0$, and Φ_L is injective, we know that $d_1 \neq 0$. We will argue that $d_i = s_1 \cdots s_{i-1}$, i.e., that L is column-major. Since

$$\Phi_L^N : [0, N) \to [0, N)$$

is a bijection, we know that 1 is in the image of Φ_L, which implies that $d_1 = 1$. Suppose $1 < i \leq m$, and suppose we have proved that $d_j = s_1 \cdots s_{j-1}$ for all $j < i$. Consider the stride d_i. We know that there is no $(x_1, \ldots, x_{i-1}, 0, \ldots, 0)$ such that $(x_1, \ldots, x_{i-1}, 0, \ldots, 0) \cdot (d_1, \ldots, d_m) = s_1 \cdots s_{i-1}$, since the largest possible value of such an expression is

$$\sum_{j=1}^{i-1}(s_j - 1)(s_1 \cdots s_{j-1}) = s_1 \cdots s_i - 1.$$

Since Φ_L is surjective, and $d_i \leq d_{i+1} \leq \cdots \leq d_m$, it follows that the next largest value of Φ_L is d_i, so we must have $d_i = s_1 \cdots s_{i-1}$, as claimed.

Returning to our main goal, consider the layout

$$\mathsf{sort}(\mathsf{squeeze}(A)) = (s'_1, \ldots, s'_{m'}) : (d'_1, \ldots, d'_{m'}).$$

Then there exist $j_1 < \cdots < j_{m'}$ such that $s'_i = s_{j_i}$ and $d'_i = d_{j_i}$ for each $1 \leq i \leq m'$. If $1 \leq i < m'$, then

$$s'_i d'_i = s_{j_i} d_{j_i} = s_{j_i} s_1 \cdots s_{j_i - 1}$$

divides

$$d'_{i+1} = s_1 \cdots s_{j_{i+1}-1}.$$

If $i = m'$, then

$$s'_{m'} d'_{m'} = s_{j_{m'}} d_{j_{m'}} = s_{j_{m'}} s_1 \cdots s_{j_{m'}-1}$$

divides

$$N = s_1 \cdots s_m.$$

We conclude that A is N-complementable. $\qquad\square$

Proposition 2.1.6.33. *Suppose N is a positive integer, and A is a N-complementable flat layout. If B is a flat layout such that*

1. *B is a N-complement of L,*

2. *B is coalesced, and*

3. *B is sorted,*

then $B = \mathsf{comp}^\flat(A, N)$.

Proof. Conditions 1 and 2 imply that Φ_B and $\Phi_{\mathsf{comp}^\flat(A,N)}$ have the same image. Since B and $\mathsf{comp}^\flat(A,N)$ are sorted, we know by Lemma 2.1.6.22 that Φ_B and $\Phi_{\mathsf{comp}^\flat(A,N)}$ are increasing. Combining these two facts, it follows that $\Phi_B = \Phi_{\mathsf{comp}^\flat(A,N)}$. Proposition 2.1.4.18 and condition 3 then imply that

$$B = \mathsf{coal}^\flat(B) = \mathsf{coal}^\flat(\mathsf{comp}^\flat(A,N)) = \mathsf{comp}^\flat(A,N).$$

$\square$

Lemma 2.1.6.34. *Suppose A is a flat layout. If $N_1 \leq N_2$ are positive integers such that A is N_1-complementable and A is N_2-complementable, then*

$$\Phi_{\mathsf{comp}^\flat(A,N_2)} \,|_{[0,N_1)} = \Phi_{\mathsf{comp}^\flat(A,N_1)}.$$

Proof. Write

$$\mathsf{sort}(\mathsf{squeeze}(A)) = (s_1, \ldots, s_m) : (d_1, \ldots, d_m),$$

$$C = \left(d_1, \frac{d_2}{s_1 d_1}, \frac{d_3}{s_2 d_2}, \ldots, \frac{d_m}{s_{m-1} d_{m-1}}\right) : \left(1, s_1 d_1, s_2 d_2, \ldots, s_{m-1} d_{m-1}\right)$$

and write

$$E_1 = \left(\frac{N_1}{s_m d_m}\right) : (s_m d_m),$$

$$E_2 = \left(\frac{N_2}{s_m d_m}\right) : (s_m d_m),$$

$$C_1 = C \star E_1,$$

$$C_2 = C \star E_2,$$

so that

$$\mathsf{comp}^\flat(A) = \mathsf{coal}^\flat(C)$$

$$\mathsf{comp}^\flat(A, N_1) = \mathsf{coal}^\flat(C_1)$$

$$\mathsf{comp}^\flat(A, N_2) = \mathsf{coal}^\flat(C_2).$$

Then we have a commuting diagram

$$
\begin{array}{ccccccc}
[0, \mathsf{size}(C_1)) & \xrightarrow{\mathsf{colex}^{-1}_{(\mathsf{size}(C), N_1)}} & [0, \mathsf{size}(C)) \times [0, N_1) & \xrightarrow{\Phi_C \times s_m d_m} & \mathbb{Z} \times \mathbb{Z} & \xrightarrow{+} & \mathbb{Z} \\
\downarrow \subseteq & & \downarrow \mathsf{id} \times \subseteq & & \downarrow \mathsf{id} & & \downarrow \mathsf{id} \\
[0, \mathsf{size}(C_2)) & \xrightarrow{\mathsf{colex}^{-1}_{(\mathsf{size}(C), N_2)}} & [0, \mathsf{size}(C)) \times [0, N_2) & \xrightarrow{\Phi_C \times s_m d_m} & \mathbb{Z} \times \mathbb{Z} & \xrightarrow{+} & \mathbb{Z}
\end{array}
$$

where, by Proposition 2.1.3.40, the composite of the top row is the layout function of $C_1 = C \star E_1$, and the composite of the bottom row is the layout function of $C_2 = C \star E_2$. This tells us that the restriction of Φ_{C_2} to $[0, \mathsf{size}(C_2))$ is Φ_{C_1}, and the result follow from the fact that

$$\Phi_{\mathsf{comp}^\flat(A,N_1)} = \Phi_{C_1}$$

$$\Phi_{\mathsf{comp}^\flat(A,N_2)} = \Phi_{C_2}.$$

$\square$

2.1.7 Further operations

In this section, we define several further operations on flat layouts, namely *composition*, *flat division*, and *flat products*. These are the flattened variants of more natural operations on (nested) layouts. We do not often work with these operations, but include them anyway for completeness.

2.1.7.1 Composition

If A and B are flat layouts, then the composite $B \circ A$ is a flat layout whose layout function is the composite of the layout functions of A and B. More precisely, we have the following definition.

Definition 2.1.7.1. Suppose A and B are flat layouts. We say the flat layout C is the *composition* of A and B, and write $C = B \circ A$, if

1. C is non-degenerate,

2. $\mathsf{shape}(A) = \mathsf{shape}(R)$,

3. $\Phi_R = \Phi_B \circ \Phi_A^{\mathsf{size}(B)}$.

Remark 2.1.7.2. Note that condition 2 in our definition ensures that Φ_R and Φ_A have the same domain, and condition 3 implies $\mathsf{cosize}(A) \le \mathsf{size}(B)$.

Example 2.1.7.3. If $A = (2,3) : (5,6)$ and $B = (80) : (10)$, then

$$B \circ A = (2,3) : (50,60).$$

More generally, if

$$A = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

is a non-degenerate flat layout, and

$$B = (t) : (e)$$

is a rank 1 flat layout with $t \ge \mathsf{cosize}(A)$, then A and B are composable, and

$$B \circ A = (s_1, \ldots, s_m) : (td_1, \ldots, td_m).$$

Example 2.1.7.4. If $A = (128, 128) : (0,0)$ and $B = (64,32) : (1,64)$, then

$$B \circ A = (128, 128) : (0,0).$$

More generally, if A is a flat layout each of whose stride entries is zero, and B is any flat layout, then A and B are composable with $B \circ A = A$.

Example 2.1.7.5. If $A = (64, 32) : (2, 256)$ and $B = (2048, 2048) : (1, 2048)$, then

$$B \circ A = (64, 32) : (2, 256).$$

More generally, if A is any flat layout, and B is a column-major flat layout with $\mathsf{cosize}(A) \le \mathsf{size}(B)$, then $B \circ A = A$.

Example 2.1.7.6. If $A = (4) : (2)$ and $B = (2,2,6) : (12,6,1)$, then there is no flat layout R with $R = B \circ A$.

Remark 2.1.7.7. If B' and B have the same layout function, then $B \circ A = B' \circ A$.

Remark 2.1.7.8. Flat layouts are a special case of the more general notion of *layouts* (Definition 2.3.1.1). It turns out that there are cases (such as Example 2.1.7.6) where there does not exist a flat layout C with $C = B \circ A$, but there does exist a (nested) layout C with $C = B \circ A$ (see Example 2.3.7.6). For this reason, we postpone further discussion and analysis of composition until we have defined layouts in their full generality.

2.1.7.2 Flat division

If A and B are flat layouts, then the flat division of A by B is a flattened version of the more natural *logical division* of layouts. See Section 2.3.8 for details.

Definition 2.1.7.9. Suppose A and B are flat layouts, and that B is $\mathsf{size}(A)$-complementable, with

$$B^c = \mathsf{comp}^\flat(B, \mathsf{size}(A)).$$

We define the *flat division* of A by B to be the flat layout

$$A \oslash^\flat B = A \circ (B \star B^c).$$

Example 2.1.7.10. If $A = (2,2,2,2) : (1,4,2,8)$ and $B = (2,2) : (4,2)$, then

$$A \oslash^\flat B = (2,2,2,2) : (4,2,1,8).$$

Example 2.1.7.11. If $A = (3,5,9,6) : (54,0,6,1)$ and $B = (6,3) : (135,1)$, then

$$A \oslash^\flat B = (6,3,5,9) : (1,54,0,6).$$

Example 2.1.7.12. If A is any flat layout and $B = () : ()$ is the empty layout, then

$$A \oslash^\flat B = A.$$

2.1.7.3 Flat products

If A and B are flat layouts, then the flat product $A \otimes^\flat B$ of A and B is a flattened version of the more natural *logical product* of layouts. See Section 2.3.9 for details.

Definition 2.1.7.13. Suppose A and B are flat layouts, and that A is $\mathsf{size}(A)\cdot\mathsf{cosize}(B)$-complementable, with

$$A^c = \mathsf{comp}^\flat(A, \mathsf{size}(A) \cdot \mathsf{cosize}(B)).$$

We define the flat product of A and B by

$$A \otimes^\flat B = A \star (A^c \circ B).$$

Example 2.1.7.14. If $A = (2,2,2) : (1,2,4)$ and $B = (2,2,2) : (1,2,4)$, then

$$A \otimes^\flat B = (2,2,2,2,2,2) : (1,2,4,8,16,32).$$

Example 2.1.7.15. If $A = (2,2,2) : (1,2,4)$ and $B = (3,5) : (5,1)$, then

$$A \otimes^\flat B = (2,2,2,3,5) : (1,2,4,40,8).$$

Example 2.1.7.16. If A is any flat layout and $B = () : ()$ is the empty layout, then

$$A \otimes^\flat B = A.$$

2.1.8 Tractable flat layouts

In this section we define an especially well-behaved class of flat layouts, called *tractable* flat layouts. Tractable flat layouts include the most important examples of interest, such as row-major, column-major, compact, and complementable layouts. Later on, we will see that tractable flat layouts are precisely the layouts which arise from a certain category **Tuple**.

Definition 2.1.8.1. Suppose L is a flat layout, and write

$$\mathsf{sort}(L) = (s_1, \ldots, s_m) : (d_1, \ldots, d_m).$$

We say L is *tractable* if for each $1 \leq i < m$, we have

1. $d_i = 0$, or

2. $s_i d_i$ divides d_{i+1}.

Example 2.1.8.2. The flat layout

$$L = (12) : (17)$$

is tractable. More generally, any flat layout of rank 1 is tractable.

Example 2.1.8.3. The flat layout

$$L = (2, 4, 32) : (1, 2, 8)$$

is tractable. More generally, any column-major layout

$$L = (s_1, \ldots, s_m) : (1, s_1, \ldots, s_1 \cdots s_{m-1})$$

is tractable.

Example 2.1.8.4. The flat layout

$$L = (2, 4, 32) : (128, 32, 1)$$

is tractable. More generally, any row-major layout

$$L = (s_1, \ldots, s_m) : (s_2 \cdots s_m, \ldots, s_m, 1)$$

is tractable.

Example 2.1.8.5. The flat layout

$$L = (3, 3, 1, 3, 3, 1, 3) : (81, 1, 0, 9, 3, 0, 27)$$

is tractable. More generally, any compact flat layout is tractable.

Example 2.1.8.6. The flat layout

$$L = (3, 7, 7) : (0, 15, 0)$$

is tractable. More generally, any flat layout with exactly one non-zero stride is tractable.

Example 2.1.8.7. The flat layout

$$L = (2, 2, 2, 2) : (1, 2048, 16, 64)$$

is tractable. More generally, any complementable flat layout is tractable.

Example 2.1.8.8. Suppose L is a flat layout. If L is tractable and $I \subset \langle m \rangle$ is any subset, then the restriction $L \mid_I$ is tractable. In particular, if L is tractable, then $\mathsf{squeeze}(L)$ and $\mathsf{filter}(L)$ are tractable.

Example 2.1.8.9. The flat layout

$$L = (4, 8) : (3, 3)$$

is not tractable. In particular, this shows that the concatenation $L_1 \star L_2$ of tractable flat layouts L_1 and L_2 need not be tractable.

Observation 2.1.8.10. If L is a tractable flat layout and no entry of $\mathsf{stride}(L)$ is equal to 0, then L is complementable. In particular, if L is tractable, then $\mathsf{filter}(L)$ is complementable.

We conclude this section by enumerating a family of equivalent conditions for a flat layout L to be tractable.

Proposition 2.1.8.11. *Suppose L is a flat layout. Then the following conditions are equivalent.*

    1. L is tractable.

    2. $\mathsf{sort}(L)$ is tractable.

    3. $\mathsf{filter}(L)$ is tractable.

    4. $\mathsf{filter}(L)$ is complementable.

Proof. Suppose L is a flat layout.

- $(1 \Leftrightarrow 2)$: This follows from the fact that

$$\mathsf{sort}(\mathsf{sort}(L)) = \mathsf{sort}(L).$$

- $(1 \Leftrightarrow 3)$: This follows from the fact that

$$\mathsf{sort}(\mathsf{filter}(L)) = \mathsf{filter}(\mathsf{sort}(L)).$$

- $(3 \Leftrightarrow 4)$: This follows from the fact that if

$$L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

  is a flat layout such that each of the stride entries d_i is nonzero, then the definition of tractability coincides with that of complementability.

$\square$

2.2 Nested Tuples

In this section, we introduce nested tuples, which are the generalization of tuples needed to define layouts in full generality.

2.2.1 Profiles

A nested tuple S is determined by its *flattening*, which is an ordinary tuple, and its *profile*, which describes parenthesization pattern on S. We define profiles precisely as follows.

Definition 2.2.1.1. A *profile* P is either

1. $P = *$, or

2. a tuple $P = (P_1, \ldots, P_r)$ of profiles $P_1, \ldots, P_r$ for some $r \geq 0$.

We write $\mathsf{Profile}$ for the set of profiles.

Example 2.2.1.2. Here are some examples of profiles.

$$
\begin{aligned}
P_1 &= (*, *) \\
P_2 &= (*, (*, *)) \\
P_3 &= ((*, *), (*, *)) \\
P_4 &= ((*, *, *), (*, ())) \\
P_5 &= () \\
P_6 &= *
\end{aligned}
$$

Let's define some important attributes of profiles.

Definition 2.2.1.3. Suppose P is a profile.

- The *rank* of X is

$$
\mathsf{rank}(P) = \begin{cases} 1 & P = * \\ r & P = (P_1, \ldots, P_r) \text{ is a tuple of profiles.} \end{cases}
$$

- The *length* of P is

$$
\mathsf{len}(P) = \begin{cases} 1 & P = * \\ \sum_{i=1}^{r} \mathsf{len}(P_i) & P = (P_1, \ldots, P_r) \text{ is a tuple of profiles.} \end{cases}
$$

- The *depth* of P is

$$
\mathsf{depth}(P) = \begin{cases} 0 & P = * \\ 1 + \max_{1 \leq i \leq r}(\mathsf{depth}(P_i)) & P = (P_1, \ldots, P_r) \text{ is a tuple of profiles.} \end{cases}
$$

Example 2.2.1.4. Here are some examples of profiles, together with their rank, length, and depth :

$$
\begin{aligned}
P &= * & \mathsf{rank}(P) &= 1, & \mathsf{len}(P) &= 1, & \mathsf{depth}(P) &= 0 \\
P &= (*, *, *) & \mathsf{rank}(P) &= 3, & \mathsf{len}(P) &= 3, & \mathsf{depth}(P) &= 1 \\
P &= (((*, *), *, *), *, *), & \mathsf{rank}(P) &= 3, & \mathsf{len}(P) &= 6, & \mathsf{depth}(P) &= 3 \\
P &= (((), ()), (*, (*, *))), & \mathsf{rank}(P) &= 2, & \mathsf{len}(P) &= 3, & \mathsf{depth}(P) &= 3
\end{aligned}
$$

Definition 2.2.1.5. Suppose P is a profile with $\mathsf{rank}(P) = r$. If $1 \le i \le r$, then the ith *mode of P* is

$$\mathsf{mode}_i(P) = \begin{cases} P & \mathsf{depth}(P) = 0 \text{ (hence } i = r = 1), \\ P_i & P = (P_1, \dots, P_r) \text{ has depth } \ge 1. \end{cases}$$

Example 2.2.1.6. If $P = ((*, *), (()), ((*, (*, *))))$ then the modes of P are

$$\mathsf{mode}_1(P) = (*, *)$$
$$\mathsf{mode}_2(P) = (())$$
$$\mathsf{mode}_3(P) = (*, (*, *)).$$

The following notation will be useful.

Notation 2.2.1.7. Suppose P is a profile of depth > 0. For any $1 \le j \le \mathsf{rank}(P)$, we write

$$\mathsf{len}_j(X) = \mathsf{len}(\mathsf{mode}_j(P)),$$
$$\mathsf{len}_{<j}(P) = \sum_{i=1}^{j-1} \mathsf{len}_i(X),$$
$$\mathsf{len}_{\le j}(X) = \mathsf{len}_{<j}(P) + \mathsf{len}_j(P)$$

The most important operation supported by profiles is *substitution*: If Q is a profile of length m, and $P_1, \dots, P_m$ are profiles, then we can obtain a new profile $(P_1, \dots, P_m)_Q$ by substituting the ith entry of Q with the profile P_i, for each $1 \le i \le m$. More precisely, we have the following definition.

Definition 2.2.1.8. Suppose Q is a profile of length m, and suppose $P_1, \dots, P_m$ are profiles. Then the *Q-substitution* of $P_1, \dots, P_m$ is the profile

$$(P_1, \dots, P_m)_Q$$

defined as follows. Write $\mathsf{depth}(Q) = d$ and $\mathsf{rank}(Q) = r$.

- If $d = 0$, then $m = 1$, and we define
$$(P_1)_Q = P_1.$$

- Suppose next that $d > 0$, and that we have defined Q'-substitution for all profiles Q' of depth $< d$. We can write
$$Q = (Q_1, \dots, Q_r)$$
where each mode $Q_i = \mathsf{mode}_i(Q)$ has depth $< d$. If for each $1 \le i \le r$, we set
$$\ell_i = \mathsf{len}(P_1) + \dots + \mathsf{len}(P_{i-1}),$$
then we define
$$(P_1, \dots, P_r)_Q = ((P_1, \dots, P_{\ell_2})_{Q_1}, \dots, (P_{\ell_r+1}, \dots, P_{\ell_{r+1}})_{Q_r}).$$

Example 2.2.1.9. If $Q = (*, *)$ and $P_1 = (*, *)$, $P_2 = (*, *, *)$,

$$(P_1, P_2)_Q = ((*, *), (*, *, *)).$$

More generally, if $Q = (*, \dots, *)$ is the profile with $\mathsf{depth}(Q) = 1$ and $\mathsf{len}(Q) = \mathsf{rank}(Q) = r$, then

$$(P_1, \dots, P_r)_Q = (P_1, \dots, P_r)$$

is ordinary concatenation.

Aside 2.2.1.10. There is an operadic interpretation of Q-substitution. The set $\mathsf{Profile}$ of profiles has the structure of a (non-symmetric) operad: the set

$$\mathsf{Profile}(n) = \{P \in \mathsf{Profile} \mid \mathsf{len}(P) = n\}$$

forms the collection of n-ary operations of $\mathsf{Profile}$, and if $n = m_1 + \cdots + m_r$, then the structure map

$$\mathsf{Profile}(m_1) \times \cdots \times \mathsf{Profile}(m_r) \times \mathsf{Profile}(n) \longrightarrow \mathsf{Profile}(m_1 + \cdots + m_r)$$

$$(P_1, \ldots, P_r), Q \longmapsto (P_1, \ldots, P_r)_Q$$

is given by Q-substitution. One can also form the cofree symmetric operad on this non-symmetric operad, which amounts to endowing the sets of n-ary operations with trivial symmetric group action.

2.2.2 Basic definitions

Having defined profiles and their basic properties, we can now define nested tuples.

Definition 2.2.2.1. If V is a set, then a *nested tuple X* with entries in V is a pair $(X^\flat, P)$ consisting of

1. a tuple $X^\flat = (x_1, \ldots, x_m)$ with entries in V, called the *flattening* of X, and

2. a profile $\mathsf{prof}(X) = P$ of length m, called the *profile* of X.

We write $\mathsf{Nest}(V)$ for the set of all nested tuples with entries in a set V.

Example 2.2.2.2. Here are some examples of nested tuples, together with their flattening and profile.

$$
\begin{aligned}
&X = (2, (2, 2)) && X^\flat = (2, 2, 2) && \mathsf{prof}(X) = (*, (*, *)) \\
&X = 25 && X^\flat = (25) && \mathsf{prof}(X) = * \\
&X = (((2, 2, 2), 8), 64) && X^\flat = (2, 2, 2, 8, 26) && \mathsf{prof}(X) = (((*, *, *), *), *) \\
&X = ((), (32, ()), (4, 8)) && X^\flat = (32, 4, 8) && \mathsf{prof}(X) = ((), (*, ()), (*, *))
\end{aligned}
$$

Notation 2.2.2.3. We sometimes write

$$X = (x_1, \ldots, x_m)_P$$

to denote a nested tuple with $X^\flat = (x_1, \ldots, x_m)$ and profile $\mathsf{prof}(X) = P$.

Observation 2.2.2.4. If V is any set, then by definition, we have a pullback square

$$
\begin{CD}
\mathsf{Nest}(V) @>{\mathsf{prof}(-)}>> \mathsf{Profile} \\
@V{(-)^\flat}VV @VV{\mathsf{len}(-)}V \\
\mathsf{Tuple}(V) @>>{\mathsf{len}(-)}> \mathbb{N}.
\end{CD}
$$

Remark 2.2.2.5. Given the recursive definition of profiles, we could equivalently define a nested tuple with entries in V to be either

1. an element of V, or

2. a tuple of nested tuples with entries in V.

Let's define some important attributes of nested tuples. Each such attribute of a nested tuple X is inhereted by its flattening $X^\flat$ or its profile $\mathsf{prof}(X)$.

Definition 2.2.2.6. Suppose X is a nested tuple with entries in V.

- The *rank* of X is
$$\mathsf{rank}(X) = \mathsf{rank}(P)$$

- The *length* of X is
$$\mathsf{len}(X) = \mathsf{len}(P) = \mathsf{len}(X^\flat)$$

- The *depth* of X is
$$\mathsf{depth}(X) = \mathsf{depth}(P)$$

- If $V = \mathbb{Z}$, then the *size* of X is
$$\mathsf{size}(X) = \mathsf{size}(X^\flat).$$

Example 2.2.2.7. Here are some examples of nested tuples of integers, together with their rank, length, depth, and size:

$$X = 27 \qquad\qquad \mathsf{rank}(X) = 1, \quad \mathsf{len}(X) = 1, \quad \mathsf{depth}(X) = 0, \quad \mathsf{size}(X) = 27$$
$$X = (2, 10, 5) \qquad\qquad \mathsf{rank}(X) = 3, \quad \mathsf{len}(X) = 3, \quad \mathsf{depth}(X) = 1, \quad \mathsf{size}(X) = 100$$
$$X = (((3, 4), 2, 2), 8, 9), \qquad \mathsf{rank}(X) = 3, \quad \mathsf{len}(X) = 6, \quad \mathsf{depth}(X) = 3, \quad \mathsf{size}(X) = 3096$$
$$X = (((), ()), (2, (5, 5))), \qquad \mathsf{rank}(X) = 2, \quad \mathsf{len}(X) = 3, \quad \mathsf{depth}(X) = 3, \quad \mathsf{size}(X) = 50$$

Example 2.2.2.8. A nested tuple of integers with depth 0 is simply an integer.

Example 2.2.2.9. A nested tuple of integers with depth 1 is simply a tuple of integers. If X is such a nested tuple, then $\mathsf{rank}(X) = \mathsf{len}(X)$.

Definition 2.2.2.10. Suppose $X = (x_1, \ldots, x_m)_P$ is a nested tuple with $\mathsf{rank}(X) = r$. If $1 \leq i \leq r$, then the ith *mode of X* to be the nested tuple

$$\mathsf{mode}_i(X) = (x_{\mathsf{len}_{<i}(P)+1}, \ldots, x_{\mathsf{len}_{\leq i}(P)})_{\mathsf{mode}_i(P)}.$$

Example 2.2.2.11. If
$$X = ((3), 4, ((10, 10), 12)),$$

then the modes of X are

$$\mathsf{mode}_1(X) = (3)$$
$$\mathsf{mode}_2(X) = 4$$
$$\mathsf{mode}_3(X) = ((10, 10), 12)$$

Example 2.2.2.12. If $X = (32, 5, 6, 64)$, then the modes of X are

$$\mathsf{mode}_1(X) = 32$$
$$\mathsf{mode}_2(X) = 5$$
$$\mathsf{mode}_3(X) = 6$$
$$\mathsf{mode}_4(X) = 64$$

It will be convenient to introduce the following notation.

Notation 2.2.2.13. Suppose X is a nested tuple of integers with $\mathsf{depth}(X) > 0$. For any $1 \leq j \leq \mathsf{rank}(X)$, we write

$$\mathsf{len}_j(X) = \mathsf{len}(\mathsf{mode}_j(X)),$$

$$\mathsf{len}_{<j}(X) = \sum_{i=1}^{j-1} \mathsf{len}_i(X),$$

$$\mathsf{len}_{\leq j}(X) = \mathsf{len}_{<j}(X) + \mathsf{len}_j(X)$$

and similarly, we write

$$\mathsf{size}_j(X) = \mathsf{size}(\mathsf{mode}_j(X)),$$

$$\mathsf{size}_{<j}(X) = \prod_{i=1}^{j-1} \mathsf{size}_j(X), \text{ and}$$

$$\mathsf{size}_{\leq j}(X) = \mathsf{size}_{<j}(X) \cdot \mathsf{size}_j(X).$$

Definition 2.2.2.14. If $X = (x_1, \ldots, x_m)_P$ is a nested tuple and $1 \leq i \leq m$, then the ith entry of X is

$$\mathsf{entry}_i(X) = \mathsf{entry}_i(X^\flat) = x_i.$$

Example 2.2.2.15. If

$$X = ((3), 4, ((10, 10), 12)),$$

then the entries of X are

$$\mathsf{entry}_1(X) = 3$$
$$\mathsf{entry}_2(X) = 4$$
$$\mathsf{entry}_3(X) = 10$$
$$\mathsf{entry}_4(X) = 10$$
$$\mathsf{entry}_5(X) = 12.$$

Example 2.2.2.16. If $X = (32, 5, 6, 64)$, then the entries of X are

$$\mathsf{entry}_1(X) = 32$$
$$\mathsf{entry}_2(X) = 5$$
$$\mathsf{entry}_3(X) = 6$$
$$\mathsf{entry}_4(X) = 4.$$

Example 2.2.2.17. If X is a nested tuple with depth 1, then $\mathsf{mode}_i(X) = \mathsf{entry}_i(X)$ for all $1 \leq i \leq \mathsf{rank}(X) = \mathsf{len}(X)$.

Observation 2.2.2.18. If X is a nested tuple of integers, then the *entries* of X are integers, while the *modes* of X are themselves nested tuples of integers.

Finally, we introduce the notion of *congruence* of nested tuples, which indicates when nested tuples have the same profile.

Definition 2.2.2.19. If X_1 and X_2 are nested tuples, we say X_1 and X_2 are *congruent*, if

$$\mathsf{prof}(X_1) = \mathsf{prof}(X_2).$$

Example 2.2.2.20. Here are some examples of nested tuples X_1 and X_2, and whether or not they are congruent

$X_1 = 27$	$X_2 = 100$	congruent
$X_1 = (2, 2)$	$X_2 = (8, 64)$	congruent
$X_1 = ((4, 8), (4, 8))$	$X_2 = ((1, 1), (5, 10))$	congruent
$X_1 = ((64, (8, 8)), (25, (5, 5)))$	$X_2 = ((2, (3, 5)), (7, (11, 13)))$	congruent
$X_1 = 27$	$X_2 = (100)$	not congruent
$X_1 = (2, 2)$	$X_2 = (8, 64, 128)$	not congruent
$X_1 = ((4, 8), (4, 8))$	$X_2 = (((1, 1), (5, 10)))$	not congruent

2.2.3   Substitution

Recall that if Q is a profile of length r and $P_1, \ldots, P_r$ are profiles, then we defined a profile

$$(P_1, \ldots, P_r)_Q$$

called the *Q-substitution* of $P_1, \ldots, P_r$. This profile is obtained from Q by replacing the ith entry of Q with the profile P_i. We can extend this to an operation on nested tuples as follows.

Definition 2.2.3.1. Suppose $X_1, \ldots, X_m$ are nested tuples with profiles $P_1, \ldots, P_m$, and suppose Q is a profile of length m. We define the Q-substitution

$$(X_1, \ldots, X_m)_Q$$

of $X_1, \ldots, X_m$ to be the nested tuple with flattening

$$(X_1, \ldots, X_m)_Q^\flat = X_1^\flat \star \cdots \star X_m^\flat$$

and profile

$$(P_1, \ldots, P_m)_Q.$$

More generally, if $X_1, \ldots, X_m$ are nested tuples and Y is a nested tuple of length m, we define

$$(X_1, \ldots, X_m)_Y = (X_1, \ldots, X_m)_{\mathsf{prof}(Y)}.$$

Example 2.2.3.2. If $(X_1, X_2, X_3) = (64, 16, 4)$ and $Q = (*, (*, *))$, then

$$(X_1, X_2, X_3)_Q = (64, (32, 4))$$

Example 2.2.3.3. If $(X_1, X_2, X_3, X_4) = ((2, 2), (3, 3), (5, 5), (7, 7))$ and $Q = ((*, *), (*, *))$, then

$$(X_1, X_2, X_3, X_4)_Q = (((2, 2), (3, 3)), ((5, 5), (7, 7))).$$

Example 2.2.3.4. If $X = (12)$ and $Q = *$, then

$$(X)_Q = 12.$$

Example 2.2.3.5. If $X_1 = 2$, $X_2 = 2$, $X_3 = (5, 5)$, and $Q = (*, *, *)$, then

$$(X_1, X_2, X_3)_Q = (2, 2, (5, 5)) = (X_1, X_2, X_3).$$

More generally, if $X_1, \ldots, X_m$ are any nested tuples and $P = (*, \ldots, *)$ then

$$(X_1, \ldots, X_m)_Q = (X_1, \ldots, X_k)$$

is the *concatenation* of $X_1, \ldots, X_m$.

Aside 2.2.3.6. There is an *operadic* interpretation of substitutions of nested tuples. The set $\mathsf{Nest}(\mathbb{Z})$ of nested tuples of integers is an *algebra* over the operad $\mathsf{Profile}$, with structure maps given by Q-substitution:

$$\mathsf{Nest}(\mathbb{Z}) \times \cdots \times \mathsf{Nest}(\mathbb{Z}) \times \mathsf{Profile}(n) \longrightarrow \mathsf{Nest}(\mathbb{Z})$$

$$(X_1, \ldots, X_m), Q \longmapsto (X_1, \ldots, X_m)_Q.$$

2.2.4 Refinement

In this section, we introduce an important relation on nested tuples called *refinement*. Intuitively, if X' and X are nested tuples of integers, we say X' refines X if X' may be obtained from X by replacing each entry of X with some nested tuple of the same size. More precisely, we have the following definition.

Definition 2.2.4.1. If X' and X are nested tuples, then we say X' *refines* X if either

1. $X = \mathsf{size}(X')$, or

2. (a) $\mathsf{depth}(X'), \mathsf{depth}(X) > 0$,
   (b) $\mathsf{rank}(X') = \mathsf{rank}(X)$, and
   (c) for each $1 \leq i \leq \mathsf{rank}(X)$, $\mathsf{mode}_i(X')$ refines $\mathsf{mode}_i(X)$.

Notation 2.2.4.2. We write

$$X' \twoheadrightarrow X$$

to indicate that X' refines X.

Example 2.2.4.3. Here are some examples of refinements of nested tuples.

$$(2, (2, 2)) \twoheadrightarrow 8$$
$$((2, 2), (3, 3), (5, 5)) \twoheadrightarrow (4, 9, 25)$$
$$(64) \twoheadrightarrow 64$$
$$(8, ((2, 2, 2), ((1, 4), (2, 2)))) \twoheadrightarrow (8, (8, 8))$$

Observation 2.2.4.4. Refinement of nested tuples is reflexive, transitive, and antisymmetric, so refinement specifies a partial ordering on the collection of nested tuples of positive integers.

If X' refines X, then we can think of X' as being obtained from X by replacing each entry x_i of X with some nested tuple X'_i of size x_i. We refer to the nested tuple X'_i as the ith mode of X' relative to X. More precisely, we have the following definition.

Construction 2.2.4.5. Suppose X is a nested tuple of integers of length m, and suppose X' refines X. For any $1 \le i \le m$, we define a nested tuple

$$X_i' = \mathsf{mode}_i(X', X),$$

called the *ith mode of X' relative to X*, by the formula

$$\mathsf{mode}_i(X', X) = \begin{cases} X' & \mathsf{depth}(X) = 0 \text{ (hence } i = \ell = 1) \\ \mathsf{mode}_{i-N}(\mathsf{mode}_j(X'), \mathsf{mode}_j(X)) & \begin{array}{l} j \text{ is the largest integer such that} \\ N := \mathsf{len}_{<j}(X) < i. \end{array} \end{cases}$$

Example 2.2.4.6. If $X = ((4, 9), (25, 36))$, and $X' = (((2, 2), (3, 3)), (25, (6, (2, 3))))$, then X' refines X and the modes of X' relative to X are

$$\mathsf{mode}_1(X', X) = (2, 2)$$
$$\mathsf{mode}_2(X', X) = (3, 3)$$
$$\mathsf{mode}_3(X', X) = 25$$
$$\mathsf{mode}_4(X', X) = (6, (2, 3)).$$

Example 2.2.4.7. If X is any nested tuple, then X refines X, and for any $1 \le i \le \mathsf{len}(X)$ we have

$$\mathsf{mode}_i(X, X) = \mathsf{entry}_i(X).$$

Example 2.2.4.8. If $X = X^\flat$ is a tuple, and X' refines X, then for any $1 \le i \le \mathsf{len}(X)$, we have

$$\mathsf{mode}_i(X', X) = \mathsf{mode}_i(X').$$

Example 2.2.4.9. If X' is a nested tuple with $\mathsf{size}(X') = N$, then X' refines N, and the only mode of X' relative to N is

$$\mathsf{mode}_1(X', N) = X'.$$

Notation 2.2.4.10. If $X' \twoheadrightarrow X$ is a refinement and $1 \le i \le \mathsf{len}(X)$, then we write

$$\mathsf{len}_i(X', X) = \mathsf{len}(\mathsf{mode}_i(X', X))$$
$$\mathsf{len}_{<i}(X', X) = \sum_{j<i} \mathsf{len}_j(X', X)$$
$$\mathsf{len}_{\le i}(X', X) = \sum_{j \le i} \mathsf{len}_j(X', X)$$

Definition 2.2.4.11. Suppose X' refines X, and write $X_i' = \mathsf{mode}_i(X', X)$. Then the *flattening of X' relative to X* is the nested tuple

$$\mathsf{flat}(X', X) = (X_1', \ldots, X_m').$$

Example 2.2.4.12. If $X' = (((2, 2), (3, 3)), ((5, 5), (7, 7)))$ and $X = ((4, 9), (25, 49))$, then

$$\mathsf{flat}(X', X) = ((2, 2), (3, 3), (5, 5), (7, 7)).$$

Example 2.2.4.13. If X is any nested tuple, then the flattening of X relative to X is

$$\mathsf{flat}(X, X) = X^\flat.$$

Example 2.2.4.14. If $X = X^\flat$ is a tuple, and X' refines X, then the flattening of X' relative to X is

$$\mathsf{flat}(X', X) = X'.$$

Example 2.2.4.15. If X' is a nested tuple with $\mathsf{size}(X') = N$, then X' refines N, and the flattening of X' relative to N is

$$\mathsf{flat}(X', N) = (N).$$

Observation 2.2.4.16. If X' refines X, then $\mathsf{flat}(X', X)$ refines $X^\flat$.

2.3 Layouts

Having developed the necessary background on nested tuples, we turn our attention to *layouts*. These are a generalization of flat layouts in which shapes and strides are allowed to be nested tuples, rather than (flat) tuples.

2.3.1 Basic definitions

Definition 2.3.1.1. A *layout* is a pair

$$L = S : D$$

consisting of a nested tuple of positive integers

$$\mathsf{shape}(L) = S$$

called the *shape* of L, and a nested tuple of non-negative integers

$$\mathsf{stride}(L) = D$$

called the *stride* of L, such that S and D are congruent.

Definition 2.3.1.2. If $L = S : D$ is a layout, then the *rank*, *length*, *depth*, *size*, and *profile* of L are defined to be the rank, length, depth, size, and profile of S, respectively.

Example 2.3.1.3. The layout $L = (3, (3, 2)) : (3, (1, 10))$ may be pictured as follows.

$$L =$$

0	1	2	10	11	12
3	4	5	13	14	15
6	7	8	16	17	18

Example 2.3.1.4. The layout $L = ((2, 2), (2, 2)) : ((1, 4), (2, 8))$ may be pictured as follows.

$$L =$$

0	2	8	10
1	3	9	11
4	6	12	14
5	7	13	15

Example 2.3.1.5. The layout

$$L = 10 : 4$$

has $\mathsf{rank}(L) = 1$, $\mathsf{len}(L) = 1$, $\mathsf{depth}(L) = 0$, $\mathsf{size}(L) = 10$, and $\mathsf{prof}(L) = *$.

Example 2.3.1.6. The layout

$$L = (7, (2, 10, 4), (3, 7)) : (1, (7, 14, 140), (560, 1680))$$

has $\mathsf{rank}(L) = 3$, $\mathsf{len}(L) = 6$, $\mathsf{depth}(L) = 2$, $\mathsf{size}(L) = 11760$, and $\mathsf{prof}(L) = (*, (*, *, *), (*, *))$.

Example 2.3.1.7. The layout

$$L = ((2, 2, 2, (2, 2))) : ((1, 0, 8, (0, 16)))$$

has $\mathsf{rank}(L) = 1$, $\mathsf{len}(L) = 5$, $\mathsf{depth}(L) = 3$, $\mathsf{size}(L) = 32$, and $\mathsf{prof}(L) = ((*, *, *, (*, *)))$.

Example 2.3.1.8. The pair

$$S : D = (2, (2, 2)) : (1, 2, 4)$$

is NOT a layout because S and D are not congruent.

Definition 2.3.1.9. If $L = S : D$ is a layout, then for any $1 \leq i \leq \mathsf{rank}(L)$ we define the ith mode of L to be the layout

$$\mathsf{mode}_i(L) = \mathsf{mode}_i(S) : \mathsf{mode}_i(D),$$

and for any $1 \leq i \leq \mathsf{len}(L)$, we define the ith entry of L to be the layout

$$\mathsf{entry}_i(L) = \mathsf{entry}_i(S) : \mathsf{entry}_i(D).$$

Example 2.3.1.10. If $L = \big((2, 2), 9\big) : \big((3, 6), 12\big)$, then the modes of L are

$$\mathsf{mode}_1(L) = (2, 2) : (3, 6)$$
$$\mathsf{mode}_2(L) = 9 : 12$$

and the entries of L are

$$\mathsf{entry}_1(L) = 2 : 3$$
$$\mathsf{entry}_2(L) = 2 : 6$$
$$\mathsf{entry}_3(L) = 9 : 12.$$

Remark 2.3.1.11. If L is a layout, then the modes of L are also layouts, and the entries of L are layouts of depth 0.

Remark 2.3.1.12. A flat layout L is precisely a layout of depth 1. On the other hand, if L is a layout, we may obtain a flat layout $L^\flat$ as follows.

Definition 2.3.1.13. If $L = S : D$ is a layout, we define the *flattening* of L to be the flat layout

$$L^\flat = S^\flat : D^\flat.$$

Example 2.3.1.14. The flattening of $L = 10 : 4$ is $L^\flat = (10) : (4)$.

Example 2.3.1.15. The flattening of

$$L = \big((2, 2, 2, (2, 2))\big) : \big((1, 0, 8, (0, 16))\big)$$

is

$$L^\flat = (2, 2, 2, 2, 2) : (1, 0, 8, 0, 16).$$

Remark 2.3.1.16. If L is a layout then $\operatorname{len}(L) = \operatorname{rank}(L^\flat)$, and for any $1 \le i \le \operatorname{len}(L)$, we have

$$\operatorname{entry}_i(L) = \operatorname{mode}_i(L^\flat).$$

We can use the flattening construction above to extend many concepts from flat layouts to nested layouts. For example:

Construction 2.3.1.17 (Layout function). If L is a nested layout, we define the layout function Φ_L of L by

$$\Phi_L = \Phi_{L^\flat},$$

where $\Phi_{L^\flat}$ is the layout function of Construction 2.1.2.19. Similarly, if N is such that $\operatorname{Image}(\Phi_L) \subset [0, N)$, we define

$$\Phi_L^N = \Phi_{L^\flat}^N$$

to be the factorization of Φ_L through the inclusion $[0, N) \subset \mathbb{Z}$.

Example 2.3.1.18. If $L = ((2,2), 2) : ((3, 0), 10)$, then the layout function

$$\Phi_L : [0, 8) \to \mathbb{Z}$$

of L is given by

$$
\Phi_L \quad
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
0 & 3 & 0 & 3 & 10 & 13 & 10 & 13
\end{array}
$$

Given a layout L, we can obtain a flat layout $L^\flat$, and a profile $P = \operatorname{prof}(L)$. Conversely, if we are given a flat layout L and a profile P with the same length as L, then we can construct a layout with flattening L and profile P as follows.

Construction 2.3.1.19. If L is a flat layout, and P is a profile with $\operatorname{len}(P) = \operatorname{len}(L)$, then we can define

$$L = L_P$$

to be the layout with shape

$$\operatorname{shape}(L) = \operatorname{shape}(L)_P$$

and stride

$$\operatorname{stride}(L) = \operatorname{stride}(L)_P$$

where $(-)_P$ is the P-substitution operation of Definition 2.2.1.8.

Example 2.3.1.20. If $L = (8, 8, 8) : (1, 64, 8)$ and $P = (*, (*, *))$, then

$$L_P = (8, (8, 8)), (1, (64, 8)).$$

Example 2.3.1.21. If $L = (128) : (2)$ and $P = *$, then

$$L_P = 128 : 2.$$

Proposition 2.3.1.22. *If L' is a flat layout and P is a profile with $\operatorname{len}(L') = \operatorname{len}(P)$, then there exists a unique layout L whose flattening is $L^\flat = L'$ and whose profile is $\operatorname{prof}(L) = P$, namely $L = L'_P$.*

Proof. This follows from the definition of nested tuples, since a nested tuple is uniquely determined by its flattening and its profile. $\qquad\square$

Observation 2.3.1.23. The previous proposition tells us that we have a pullback square

$$
\begin{array}{ccc}
\mathsf{Layout} & \xrightarrow{\ \mathsf{prof}(-)\ } & \mathsf{Profile} \\
{\scriptstyle (-)^{\flat}}\Big\downarrow & \lrcorner & \Big\downarrow{\scriptstyle \mathsf{len}(-)} \\
\mathsf{FlatLayout} & \xrightarrow[\ \mathsf{len}(-)\ ]{} & \mathbb{N}
\end{array}
$$

We can extend the notion of non-degeneracy to the nested case as follows.

Definition 2.3.1.24. Suppose L is a layout. We say L is *non-degenerate* if for all $1 \leq i \leq \mathsf{len}(L)$, the following condition holds:

$$
\mathsf{entry}_i(\mathsf{shape}(L)) \quad \Rightarrow \quad \mathsf{entry}_i(\mathsf{stride}(L))
$$

Example 2.3.1.25. The layouts

$$
\begin{aligned}
L_1 &= ((2,2),1) : ((1,2),0) \\
L_2 &= ((8,8),(1,16)) : ((2,32),(0,128))
\end{aligned}
$$

are non-degenerate, while the layouts

$$
\begin{aligned}
L_3 &= ((2,2),1) : ((1,2),4) \\
L_4 &= ((8,8),(1,16)) : ((2,32),(1024,128))
\end{aligned}
$$

are degenerate.

2.3.2 Basic operations

Having established the basic vocabulary for layouts, we turn to the operations they support. In this section, we define basic operations that will be needed to construct more sophisticated operations such as *coalesce*, *complement*, *composition*, *logical division*, and *logical product*.

2.3.2.1 Flattening

If L is a layout, then we may obtain a flat layout $L^{\flat}$ by flattening the shape and stride of L.

Definition 2.3.2.1. If $L = S : D$ is a layout, we define the *flattening* of L to be the flat layout

$$
L^{\flat} = S^{\flat} : D^{\flat}.
$$

Example 2.3.2.2. The flattening of

$$
L = ((2,2,2,(2,2))) : ((1,0,8,(0,16)))
$$

is

$$
L^{\flat} = (2,2,2,2,2) : (1,0,8,0,16).
$$

Example 2.3.2.3. The flattening of $L = 10 : 4$ is $L^{\flat} = (10) : (4)$.

Example 2.3.2.4. Suppose L is a layout. Then $\mathsf{depth}(L) = 1$ if and only if $L = L^{\flat}$.

2.3.2.2 Concatenate

We can concatenate layouts by concatenating their shapes and concatenating their strides.

Definition 2.3.2.5. If $L = S : D$ and $L' = S' : D'$ are layouts, then the *concatenation* of L and L' is the layout (L, L')

$$(L, L') = (S, S') : (D, D').$$

More generally, if $L_1, \ldots, L_k$ is any finite collection of layouts, with $L_i = S_i : D_i$, then the concatenation of $L_1, \ldots, L_k$ is the layout

$$(L_1, \ldots, L_k) = (S_1, \ldots, S_k) : (D_1, \ldots, D_k).$$

Remark 2.3.2.6. Concatenation of nested tuples (and hence of layouts) is not associative. For example, take $L_1 = 3 : 4$, $L_2 = 2 : 2$, and $L_3 = 5 : 1$. Then

$$\big(L_1, (L_2, L_3)\big) = \big(3, (2, 5)\big) : \big(4, (2, 1)\big) \neq \big((3, 2), 5\big) : \big((4, 2), 1\big) = \big((L_1, L_2), L_3\big).$$

Moreover, neither of these layouts is equal to the "three-fold" concatenation $(L_1, L_2, L_3) = (3, 2, 5) : (4, 2, 1)$. However, we see that each of these layouts has the same flattening, so each of these layouts has the same layout function.

Example 2.3.2.7. If $L = (3, 7, 2) : (1, 3, 6)$ and $L' = (2, (2, (4, 3))) : (5, 3, (2, 2))$, then

$$(L, L') = ((3, 7, 2), (2, (2, (4, 3)))) : ((1, 3, 6), (5, (3, (2, 2))))$$

Remark 2.3.2.8. Concatenation increases the depth of layouts. More precisely, we have

$$\mathsf{depth}(L, L') = 1 + \max(\mathsf{depth}(L), \mathsf{depth}(L')).$$

Remark 2.3.2.9. When L and L' are flat layouts, the concatenation of Definition 2.3.2.5 does *NOT* agree with the concatenation of flat layouts of Definition 2.1.3.36. Instead, these operations are related by the formula

$$L \star L' = (L, L')^\flat.$$

Remark 2.3.2.10. If L is any layout with $\mathsf{depth}(L) > 0$ and $\mathsf{rank}(L) = r$, then we may write

$$L = (\mathsf{mode}_1(L), \ldots, \mathsf{mode}_r(L))$$

as the concatenation of its modes.

Example 2.3.2.11. If

$$L = ((5, (7, 7)), 2, (4, 5)) : ((1, (35, 5)), 0, (1, 8))$$

then $L = (L_1, L_2, L_3)$ where

$$
\begin{aligned}
L_1 &= \big(5, (7, 7)\big) : \big(1, (35, 5)\big), \\
L_2 &= 2 : 0, \text{ and} \\
L_3 &= (4, 5) : (1, 8).
\end{aligned}
$$

2.3.2.3 Substitution

Recall that if $X_1, \ldots, X_k$ are nested tuples and P is a profile with $\mathsf{len}(P) = k$, then we may form the P-substitution

$$(X_1, \ldots, X_k)_P$$

which is obtained by replacing the $i$the entry of P with the nested tuple X_i. We can extend this construction from nested tuples to layouts as follows.

Definition 2.3.2.12. Suppose $L = S : D$ is a layout, and suppose P is a profile with $\mathsf{len}(P) = \mathsf{rank}(L)$. We define

$$L_P = S_P : D_P$$

where S_P and D_P are the P-substitutions of (the modes of) S and D.

Example 2.3.2.13. If $P = (*, (*, *))$ and $L = (8, 8, 8) : (1, 8, 64)$, then

$$L_P = (8, (8, 8)) : (1, (8, 64)).$$

Example 2.3.2.14. If $P = (*, (*, *)))$ and

$$L = ((2, 2), (3, 3), (5, 5)) : ((2, 1), (12, 4), (180, 36)),$$

then

$$L_P = ((2, 2), ((3, 3), (5, 5))) : ((2, 1), ((12, 4), (180, 36))).$$

Example 2.3.2.15. If $L = (16) : (1)$ and $P = *$, then

$$L_P = 16 : 1.$$

2.3.3 Coalesce

Recall that if L is a flat layout, then $\mathsf{coal}^\flat(L)$ is a the unique flat layout of minimal rank whose layout function is Φ_L. We can make a similar construction in the setting of arbitrary (nested) layouts. We begin by defining the notion of a coalesced layout.

Definition 2.3.3.1. Suppose L is a layout. We say L is *coalesced* if one of the following conditions holds.

1. $L = 1 : 0$,

2. $\mathsf{depth}(L) = 0$ and $\mathsf{shape}(L) > 1$, or

3. $\mathsf{depth}(L) = 1$, $\mathsf{rank}(L) > 1$, and L is coalesced in the sense of Definition 2.1.4.1.

Example 2.3.3.2. The layout

$$L = (2, (2, 2)) : (1, (16, 512))$$

is not coalesced since $\mathsf{depth}(L) > 1$.

Example 2.3.3.3. The layout

$$L = (64) : (2)$$

is not coalesced, while the layout

$$L' = 64 : 2$$

is coalesced.

Example 2.3.3.4. The layout

$$L = 1 : 8$$

is not coalesced, while the layout

$$L' = 1 : 0$$

is coalesced.

Example 2.3.3.5. The empty layout

$$E = () : ()$$

is not coalesced.

Observation 2.3.3.6. Recall that a layout L is non-degenerate if

$$\mathsf{entry}_i(\mathsf{shape}(L)) = 1 \quad \Rightarrow \quad \mathsf{entry}_i(\mathsf{stride}(L)) = 0.$$

If L is coalesced, then L is non-degenerate.

If L is any layout, we can obtain a coalesced layout $\mathsf{coal}(L)$ as follows.

Construction 2.3.3.7. Suppose L is a layout, and write

$$\mathsf{coal}^\flat(L^\flat) = (s_1, \ldots, s_m) : (d_1, \ldots, d_m).$$

1. If $m > 1$, we define

$$\mathsf{coal}(L) = \mathsf{coal}^\flat(L^\flat)$$

2. If $m = 1$, we define

$$\mathsf{coal}(L) = s_1 : d_1$$

3. If $m = 0$, we define

$$\mathsf{coal}(L) = 1 : 0.$$

Example 2.3.3.8. If $E = () : ()$ is the empty layout, then

$$\mathsf{coal}(E) = 1 : 0.$$

Example 2.3.3.9. If $L = (1, 1) : (2, 4)$, then

$$\mathsf{coal}(L) = 1 : 0.$$

Example 2.3.3.10. If $L = (512) : (4)$, then

$$\mathsf{coal}(L) = 512 : 4.$$

Example 2.3.3.11. If $L = (2, 2, 2) : (1, 2, 4)$, then

$$\mathsf{coal}(L) = 8 : 1.$$

Example 2.3.3.12. If $L = ((2, 2, 2), (5, 5)) : ((1, 2, 4), (10, 50))$, then

$$\mathsf{coal}(L) = (8, 25) : (1, 10).$$

Remark 2.3.3.13. If L is a layout, then $\mathsf{coal}(L)$ has depth 0 or 1.

Proposition 2.3.3.14. *If A and B are layouts, then*

$$\Phi_A = \Phi_B \quad \Leftrightarrow \quad \mathsf{coal}(A) = \mathsf{coal}(B).$$

Proof. Using Proposition 2.1.4.18, we have

$$\begin{aligned}
\Phi_A = \Phi_B \quad &\Leftrightarrow \quad \Phi_{A^\flat} = \Phi_{B^\flat} \\
&\Leftrightarrow \quad \mathsf{coal}^\flat(A^\flat) = \mathsf{coal}^\flat(B^\flat) \\
&\Leftrightarrow \quad \mathsf{coal}(A) = \mathsf{coal}(B).
\end{aligned}$$

$\square$

Definition 2.3.3.15. If L is a layout, define the *complexity* of L to be the integer

$$\mathsf{complexity}(L) = \mathsf{len}(L) + \mathsf{depth}(L).$$

Proposition 2.3.3.16. *If L is a layout and $\mathsf{size}(L) > 1$, then $\mathsf{coal}(L)$ is the unique complexity minimizing layout whose layout function is Φ_L.*

Proof. Suppose L' is a layout with the same layout function as L, and suppose $\mathsf{coal}(L') \neq 1 : 0$. Then

$$\mathsf{len}(L') \geq \mathsf{len}(\mathsf{coal}(L')) = \mathsf{len}(\mathsf{coal}(L)).$$

There are two cases to consider.

- (Case 1): Suppose $\mathsf{len}(L') > 1$. Then $\mathsf{depth}(L') \geq 1 \geq \mathsf{depth}(\mathsf{coal}(L))$. Combining these inequalities, we observe that

$$\mathsf{complexity}(L') \geq \mathsf{complexity}(\mathsf{coal}(L)),$$

  where equality holds if and only if $L' = \mathsf{coal}(L') = \mathsf{coal}(L)$.

- (Case 2): Suppose $\mathsf{len}(L') = 1$. Then $L' = (s) : (d)$ or $L' = s : d$ for some integers $s > 1$ and $d \geq 0$. In either case, we have $\mathsf{coal}(L') = s : d$, and

$$\mathsf{complexity}(L') \geq \mathsf{complexity}(\mathsf{coal}(L)),$$

  where equality holds if and only if $L' = s : d = \mathsf{coal}(L)$.

$\square$

Remark 2.3.3.17. The only reason that we need to exclude the case $\mathsf{size}(L) = 1$ is that if $\mathsf{size}(L) = 1$, then $1 : 0$ and the empty layout $() : ()$ are distinct layouts with minimal complexity, and the same layout function as L (namely the trivial layout function $0 \mapsto 0$).

2.3.4 Relative coalesce

There is an important invariant of coalesce called *relative coalesce*, denoted $\mathsf{coal}(L, \bar{S})$. This operation receives as an additional input a nested tuple $\bar{S}$ which is refined by $\mathsf{shape}(L)$. In this case, the relative coalesce operation simplifies the layout L has much as possible, while ensuring that the resulting shape still refines $\bar{S}$.

Definition 2.3.4.1. Suppose $L = S : D$ is a layout, and suppose $\bar{S}$ is some nested tuple of length m which is refined by S. Recall that for any $1 \leq i \leq m$, we may consider the ith mode of S relative to $\bar{S}$, denoted

$$\mathsf{mode}_i(S, \bar{S}).$$

Since S and D are congruent, there is a nested tuple

$$\mathsf{mode}_i(D, \bar{S})$$

corresponding to $\mathsf{mode}_i(S, \bar{S})$, and we define the *$i$th mode of L relative to $\bar{S}$* to be the layout

$$\mathsf{mode}_i(L, \bar{S}) = \mathsf{mode}_i(S, \bar{S}) : \mathsf{mode}_i(D, \bar{S}).$$

Example 2.3.4.2. If $\bar{S} = (4, (9, 25))$ and

$$L = ((2, 2), ((3, 3), (5, (1, 5)))) : ((1, 2), ((6, 18), (90, (0, 450))))$$

then

$$\mathsf{mode}_1(L, \bar{S}) = (2, 2) : (1, 2)$$
$$\mathsf{mode}_2(L, \bar{S}) = (3, 3) : (6, 18)$$
$$\mathsf{mode}_3(L, \bar{S}) = (5, (1, 5)) : (90, (0, 450)).$$

Observation 2.3.4.3. Suppose $L = S : D$ is a layout, and suppose $\bar{S}$ is a nested tuple of length m and profile P which is refined by S. If for any $1 \leq i \leq m$, we write

$$L_i = \mathsf{mode}_i(L, \bar{S}),$$

then

$$L = (L_1, \ldots, L_m)_P$$

is the P-substitution of its relative modes.

Definition 2.3.4.4. Suppose $L = S : D$ is a layout, and suppose $\bar{S}$ is a nested tuple of length m and profile P which is refined by S. We say L is *coalesced over $\bar{S}$* if each relative mode

$$\mathsf{mode}_i(L, \bar{S})$$

is coalesced.

Observation 2.3.4.5. In the setting of Definition 2.3.4.4, if L is coalesced over $\bar{S}$, then L is non-degenerate.

Example 2.3.4.6. If L is a layout, then L is coalesced over $\mathsf{shape}(L)$ if and only if L is non-degenerate, i.e.

$$\mathsf{entry}_i(\mathsf{shape}(L)) = 1 \quad \Rightarrow \quad \mathsf{entry}_i(\mathsf{stride}(L)) = 0.$$

Definition 2.3.4.7 (Relative coalesce)**.** Suppose $L = S : D$ is a layout, and suppose $\bar{S}$ is a nested tuple of length m and profile P which is refined by S. We define

$$\mathsf{coal}(L, \bar{S}) = (\mathsf{coal}(L_1), \ldots, \mathsf{coal}(L_m))_P.$$

Remark 2.3.4.8. In the setting of Definition 2.3.4.7, the shape of $\mathsf{coal}(L, \bar{S})$ refines $\bar{S}$.

Lemma 2.3.4.9. *If $L = S : D$ is a layout and S refines $\bar{S}$, then*

$$\Phi_{\mathsf{coal}(L,\bar{S})} = \Phi_L.$$

Proof. As above, let

$$L_i = \mathsf{mode}_i(L, S)$$

denote the ith mode of L relative to S, and set $\bar{L}_i = \mathsf{coal}(L_i)$. Then

$$
\begin{aligned}
\Phi_{\mathsf{coal}(L,\bar{S})} &= \Phi_{(\bar{L}_1,\ldots,\bar{L}_m)_{\bar{S}}} \\
&= \Phi_{(\bar{L}_1,\ldots,\bar{L}_m)} \\
&= \Phi_{\mathsf{coal}((\bar{L}_1,\ldots,\bar{L}_m))} \\
&= \Phi_{\mathsf{coal}((L_1,\ldots,L_m))} \\
&= \Phi_{(L_1,\ldots,L_m)} \\
&= \Phi_{(L_1,\ldots,L_m)_{\bar{S}}} \\
&= \Phi_L.
\end{aligned}
$$

$\square$

Proposition 2.3.4.10. *Suppose A and B are layouts, and suppose $\bar{S}$ is a nested tuple of length m such that $\mathsf{shape}(A)$ refines $\bar{S}$ and $\mathsf{shape}(B)$ refines $\bar{S}$. Then*

$$\Phi_A = \Phi_B \quad \Leftrightarrow \quad \mathsf{coal}(A, \bar{S}) = \mathsf{coal}(B, \bar{S})$$

Proof. If $\mathsf{coal}(A, \bar{S}) = \mathsf{coal}(B, \bar{S})$, the using Lemma 2.3.4.9, we have

$$\Phi_A = \Phi_{\mathsf{coal}(A,\bar{S})} = \Phi_{\mathsf{coal}(B,\bar{S})} = \Phi_B.$$

Conversely, suppose that $\Phi_A = \Phi_B$. We will argue that $\mathsf{coal}(A, \bar{S}) = \mathsf{coal}(B, \bar{S})$. Set $P = \mathsf{prof}(\bar{S})$, and for any $1 \leq i \leq m$, set

$$
\begin{aligned}
A_i &= \mathsf{mode}_i(A, \bar{S}) \\
B_i &= \mathsf{mode}_i(B, \bar{S}).
\end{aligned}
$$

Since

$$\mathsf{coal}(A, \bar{S}) = (\mathsf{coal}(A_1), \ldots, \mathsf{coal}(A_m))_P$$

and

$$\mathsf{coal}(B, \bar{S}) = (\mathsf{coal}(B_1), \ldots, \mathsf{coal}(B_m))_P$$

it suffices to prove that $\mathsf{coal}(A_i) = \mathsf{coal}(B_i)$ for all $1 \leq i \leq m$. By the associativity of colexicographic isomorphisms, we can write the layout function Φ_A of A as

$$[0, \mathsf{size}(A)) \xrightarrow{\ \mathsf{colex}^{-1}\ } \prod_{j=1}^m [0, \mathsf{size}(A_j)) \xrightarrow{\ \prod \Phi_{A_j}\ } \prod_{j=1}^m \mathbb{Z} \xrightarrow{\ + \ } \mathbb{Z}$$

and we can write the layout function Φ_B of B as

$$[0, \mathsf{size}(B)) \xrightarrow{\ \mathsf{colex}^{-1}\ } \prod_{j=1}^m [0, \mathsf{size}(B_j)) \xrightarrow{\ \prod \Phi_{B_j}\ } \prod_{j=1}^m \mathbb{Z} \xrightarrow{\ + \ } \mathbb{Z}$$

75

For a fixed $1 \leq i \leq m$, consider the subset

$$[0, \mathsf{size}(A_i)) \subset \prod_{j=1}^{m} [0, \mathsf{size}(A_j))$$

and its image

$$\mathsf{colex}([0, \mathsf{size}(A_i))) \subset [0, \mathsf{size}(A)).$$

Since $\mathsf{size}(A_j) = \mathsf{size}(B_j)$ for all $1 \leq j \leq m$, this is the same as the image

$$\mathsf{colex}([0, \mathsf{size}(B_j))) \subset [0, \mathsf{size}(B)) = [0, \mathsf{size}(B)).$$

The restriction of Φ_A to this subset is Φ_{A_i}, and the restriction of B to this subset is Φ_{B_i}, so it follows that $\Phi_{A_i} = \Phi_{B_i}$, and by Proposition 2.3.3.14, we have $\mathsf{coal}(A_i) = \mathsf{coal}(B_i)$. We deduce that

$$\mathsf{coal}(A, \bar{S}) = \mathsf{coal}(B, \bar{S}),$$

as desired. $\qquad \square$

2.3.5 Compact layouts

We can easily extend the concept of compact layouts to the nested case. Again, in terms of the standard grid diagrams depicting layouts, a layout L is compact if each integer $0 \leq i < \mathsf{size}(L)$ appears exactly once. More preciesly, we have the following definition.

Definition 2.3.5.1. Suppose L is a layout. We say L is *compact* if the layout function

$$\Phi_L^{\mathsf{cosize}(L)} : [0, \mathsf{size}(L)) \to [0, \mathsf{cosize}(L))$$

is an isomorphism.

Example 2.3.5.2. The layout

$$A = ((2,2),(2,2)) : ((1,4),(2,8)) =$$

0	2	8	10
1	3	9	11
4	6	12	14
5	7	13	15

is compact, while the layouts

$$B = ((2,2),(2,2)) : ((1,4),(2,32)) =$$

0	2	32	34
1	3	33	35
4	6	36	38
5	7	37	39

and

$$C = ((2,2),(2,2)) : ((1,4),(2,0)) =$$

0	2	0	2
1	3	1	3
4	6	4	6
5	7	5	7

are not compact.

Example 2.3.5.3. The following layouts are compact:

$$L_1 = (2,(2,2)) : (8,(1,4))$$
$$L_2 = ((8,1),(8,32)) : ((2,0),(16,128))$$
$$L_3 = 64 : 1$$

Example 2.3.5.4. The layout
$$L = (2,(2,2)) : (4,(8,16))$$
is not compact since the integer $1 \in [0,29) = [0,\mathsf{cosize}(L))$ is not in the image of Φ_L. More generally, if $\mathsf{size}(L) \neq \mathsf{cosize}(L)$, then L is not compact.

We conclude this section by listing some equivalent conditions for a layout L to be compact.

Proposition 2.3.5.5. *Suppose L is a layout. Then the following conditions are equivalent.*

1. *L is compact.*

2. *$L^\flat$ is compact.*

3. *$\mathsf{coal}(L)$ is compact.*

Proof. The equivalence of these conditions follows from the fact that

$$\Phi_L = \Phi_{L^\flat} = \Phi_{\mathsf{coal}(L)}.$$

$\square$

2.3.6 Complements

We can easily extend the concept of complement to the nested case as follows.

Definition 2.3.6.1. Suppose A and B are layouts. We say B is a *complement of* A, and write $A \perp B$, if the concatenated layout (A,B) is compact.

Lemma 2.3.6.2. *Suppose A and B are layouts. Then*

$$A \perp B \quad \Leftrightarrow \quad A^\flat \perp B^\flat.$$

Proof. This follows from the observation that $(A,B)^\flat = A^\flat \star B^\flat$. $\square$

Definition 2.3.6.3. Suppose A is a layout. We say A is *complementable* if $A^\flat$ is complementable.

Lemma 2.3.6.4. *Suppose A is a layout. Then there exists a complement B of A if and only if A is complementable.*

Proof. If A is complementable, then $A^\flat$ is complementable, so there exists a flat layout B such that the flat concatenation $A^\flat \star B$ is compact. It follows that the concatenation (A, B) is also compact, so A admits a complement. Conversely, suppose there exists a layout B such that (A, B) is compact. Then $B^\flat$ is a complement of $A^\flat$, so by Proposition 2.1.6.21, $A^\flat$ is complementable, hence, by definition, so is A. $\qquad\square$

Definition 2.3.6.5. Suppose A is a layout. If A is complementable, then we define

$$\mathsf{comp}(A) = \mathsf{coal}(\mathsf{comp}^\flat(A^\flat)),$$

as in Construction 2.1.6.16. If N is a positive integer and A is N-complementable, then we define

$$\mathsf{comp}(A, N) = \mathsf{coal}(\mathsf{comp}^\flat(A^\flat, N))$$

as in Construction 2.1.6.29.

Remark 2.3.6.6. Suppose A is a complementable layout. Then we almost always have $\mathsf{comp}(A) = \mathsf{comp}^\flat(A^\flat)$. More precisely, if $\mathsf{comp}^\flat(A^\flat)$ has length > 1, then

$$\mathsf{comp}(A) = \mathsf{comp}^\flat(A^\flat),$$

if $\mathsf{comp}^\flat(A^\flat) = (s) : (d)$ has length 1, then

$$\mathsf{comp}(A) = s : d,$$

and if $\mathsf{comp}^\flat(A^\flat) = () : ()$, then

$$\mathsf{comp}(A) = 1 : 0.$$

Definition 2.3.6.7. Suppose A is a layout and N is a positive integer. We say a layout B is a *N-complement* of A if $A \perp B$, and

$$\mathsf{size}(A) \cdot \mathsf{size}(B) = N.$$

Definition 2.3.6.8. Suppose A is a layout and N is a positive integer. We say A is *N-complementable* if the flat layout $A^\flat$ is N-complementable, as in Definition 2.1.6.24.

Proposition 2.3.6.9. *Suppose A is a layout. Then there exists a N-complement of A if and only if A is N-complementable.*

Proof. If B is a N-complement of A, then $B^\flat$ is a N-complement $A^\flat$, and so by Proposition 2.1.6.32, $A^\flat$ is N-complementable, hence, so is A. Conversely, if A is N-complementable, then $\mathsf{comp}(A, N)$ is a N-complement of A. $\qquad\square$

Example 2.3.6.10. If $A = ((4, 2), (2, 2)) : ((3, 24), (192, 96))$ and $N = 768$ then

$$\mathsf{comp}(A, N) = (3, 2, 2, 2) : (1, 12, 48, 384).$$

Example 2.3.6.11. If $A = ((16, 4), 64) : ((1, 16), 64)$ and $N = 4096$ then

$$\mathsf{comp}(A, N) = \mathsf{coal}(() : ())$$
$$= 1 : 0.$$

Example 2.3.6.12. If $A = ((16, 4), 64) : ((1, 16), 64)$ and $N = 8192$ then

$$\mathsf{comp}(A, N) = \mathsf{coal}((2) : (4096))$$
$$= 2 : 4096.$$

Example 2.3.6.13. If $A = ((16, 4), 64) : ((8, 1), 128)$ and $N = 16384$, then

$$\mathsf{comp}(A, N) = \mathsf{coal}((2, 2) : (4, 8192))$$
$$= (2, 2) : (4, 8192).$$

2.3.7 Composition

In this section, we discuss the most important operation on layouts, namely *composition*. If A and B are layouts, then the composition of A and B is a layout $B \circ A$ whose layout function is the composite of the layout functions of A and B. More precisely, we have the following definition.

Definition 2.3.7.1 (Composition of layouts)**.** Suppose A and B are layouts. The *composite* of A and B is the unique layout $B \circ A$ satisfying the following properties.

1. $\mathsf{shape}(B \circ A)$ refines $\mathsf{shape}(A)$,

2. $B \circ A$ is coalesced over $\mathsf{shape}(A)$, and

3. $\Phi_{B \circ A} = \Phi_B \circ \Phi_A^{\mathsf{size}(B)}$.

Remark 2.3.7.2. In order for $B \circ A$ to exist, we must have

$$\mathsf{Image}(\Phi_A) \subseteq [0, \mathsf{size}(B)).$$

Remark 2.3.7.3. There is an implicit assertion in the definition of layout composition, namely that there is at most one layout satisfying the three conditions. This is justified by Proposition 2.3.4.10. We might define a *weak composite* of A and B to be a layout C satisfying conditions 1. and 3. (but not necessarily 2.), in which case

$$B \circ A = \mathsf{coal}(C, \mathsf{shape}(A))$$

We will see later on that when attempting to compute compositions of layouts, it is useful to compute any weak composite C of A and B, then coalesce over $\mathsf{shape}(A)$ to form the actual composite $B \circ A$.

Remark 2.3.7.4. Note that, by Observation 2.3.4.5, condition 2. in the definition of composition implies that $B \circ A$ is non-degenerate.

Example 2.3.7.5. If $A = (3, 5) : (10, 2)$ and $B = (100) : (7)$, then

$$B \circ A = (3, 5) : (70, 14).$$

Example 2.3.7.6. If $A = (4) : (2)$ and $B = (2, 2, 6) : (12, 6, 1)$, then the composition of A and B is

$$B \circ A = ((2, 2)) : ((6, 1)).$$

Remark 2.3.7.7. Example 2.3.7.6 illustrates the fact that the composition of flat layouts A and B need not be flat.

Example 2.3.7.8. If $A = ((2, 4), 8) : ((4, 8), 8)$ and $B = (4, 4, 4, 4) : (2, 4, 8, 16)$, then

$$B \circ A = ((2, (2, 2)), (2, 4)) : ((4, (8, 8)), (8, 8)).$$

Example 2.3.7.9. If $A = ((3,(2,2)),24) : ((3,(9,18)),72)$ and $B = (9,8,3,8) : (24,3,1,384)$ then

$$B \circ A = ((3,(2,2)),(3,8)) : ((72,(3,6)),(1,384))$$

Next, we develop some useful properties for computing the composition of layouts.

Proposition 2.3.7.10. *Suppose A is a layout, and suppose B and $\tilde{B}$ are layouts such that*

- $\mathsf{size}(B) \leq \mathsf{size}(\tilde{B})$, *and*

- $\Phi_{\tilde{B}} \,|_{\mathsf{size}(B)} = \Phi_B$.

If A and B are composable, then

$$B \circ A = \tilde{B} \circ A.$$

Proof. Suppose A and B are composable. Then $\mathsf{cosize}(A) \leq \mathsf{size}(B)$, and the fact that $B \circ A$ is the composite of A and $\tilde{B}$ follows from the equality

$$\Phi_{\tilde{B}} \circ \Phi_A^{\mathsf{size}(\tilde{B})} = (\Phi_{\tilde{B}}) \,|_{\mathsf{size}(B)} \circ \Phi_A^{\mathsf{size}(B)}$$
$$= \Phi_B \circ \Phi_A^{\mathsf{size}(B)}.$$

$\square$

Corollary 2.3.7.11. *If A and B are layouts, then A and B are composable if and only if A and $\mathsf{coal}(B)$ are composable, and*

$$B \circ A = \mathsf{coal}(B) \circ A.$$

Now that we have developed the basic properties of layout composition, we turn our attention to the two most important instances of composition, namely *logical division* and *logical products*.

2.3.8 Logical division

In this section, we define the **logical division** of layouts. As a motivating example, consider the layout

$$A = (4,8) : (1,4) =$$

0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

For various purposes, we may want to *tile* the layout A. For example, here are the tilings of A by various layouts B.

When working with such tiled layouts, we would like to index into our layout with coordinates of the form (`tile_coordinate`, `tile`) where `tile` specifies which tile we are working with, and `tile_coordinate` specifies a coordinate within the specified tile. For example, if both A and B have rank 2, we would like to write $((i, j), (k, \ell))$ as the index of the (i, j)th entry of the (k, ℓ)th tile of A. The logical division of $A \oslash B$ is precisely the layout which affords us this ability.

Definition 2.3.8.1. Suppose A and B are layouts, and suppose

$$B^c = \mathsf{comp}(B, \mathsf{size}(A))$$

is the complement of B with respect to $\mathsf{size}(A)$. Then the *logical division* of A by B is the layout

$$A \oslash B = A \circ (B, B^c)$$
$$= (A \circ B, A \circ B^c).$$

Example 2.3.8.2. If $A = (4, 8) : (1, 4)$ and $B = (2, 2) : (1, 4)$, then

$$A \oslash B = ((2, 2), (2, 4)) : ((1, 4), (2, 8)),$$

81

as depicted below.

$$A = \begin{array}{|c|c|c|c|c|c|c|c|}\hline 0 & 4 & 8 & 12 & 16 & 20 & 24 & 28 \\\hline 1 & 5 & 9 & 13 & 17 & 21 & 25 & 29 \\\hline 2 & 6 & 10 & 14 & 18 & 22 & 26 & 30 \\\hline 3 & 7 & 11 & 15 & 19 & 23 & 27 & 31 \\\hline\end{array} \qquad B = \begin{array}{|c|c|}\hline 0 & 4 \\\hline 1 & 5 \\\hline\end{array}$$

$$A \oslash B = \begin{array}{|c|c|c|c|c|c|c|c|}\hline 0 & 2 & 8 & 10 & 16 & 18 & 24 & 26 \\\hline 1 & 3 & 9 & 11 & 17 & 19 & 25 & 27 \\\hline 4 & 6 & 12 & 14 & 20 & 22 & 28 & 30 \\\hline 5 & 7 & 13 & 15 & 21 & 23 & 29 & 31 \\\hline\end{array}$$

Remark 2.3.8.3. The *color* of each entry in $A \oslash B$ indicates the tile to which it belongs, and the *opacity* of each entry in $A \oslash B$ indicates which entry of the tile it represents. This is why each column of $A \oslash B$ has the same color, and each row of $A \oslash B$ has the same opacity.

Example 2.3.8.4. If $A = (4, 8) : (1, 4)$ and $B = (2, 2) : (4, 1)$, then

$$A \oslash B = ((2, 2), (2, 4)) : ((4, 1), (2, 8)),$$

as depicted below.

$$A = \begin{array}{|c|c|c|c|c|c|c|c|}\hline 0 & 4 & 8 & 12 & 16 & 20 & 24 & 28 \\\hline 1 & 5 & 9 & 13 & 17 & 21 & 25 & 29 \\\hline 2 & 6 & 10 & 14 & 18 & 22 & 26 & 30 \\\hline 3 & 7 & 11 & 15 & 19 & 23 & 27 & 31 \\\hline\end{array} \qquad B = \begin{array}{|c|c|}\hline 0 & 1 \\\hline 4 & 5 \\\hline\end{array}$$

$$A \oslash B = \begin{array}{|c|c|c|c|c|c|c|c|}\hline 0 & 2 & 8 & 10 & 16 & 18 & 24 & 26 \\\hline 4 & 6 & 12 & 14 & 20 & 22 & 28 & 30 \\\hline 1 & 3 & 9 & 11 & 17 & 19 & 25 & 27 \\\hline 5 & 7 & 13 & 15 & 21 & 23 & 29 & 31 \\\hline\end{array}$$

Remark 2.3.8.5. Note the difference between the previous two examples. The tiling of A in each of the two examples is identical, but the layout of each tile is different. In first example, the tiles have column-major layouts, while in the second example, the tiles have row-major layouts. This results in different layouts when one performs logical division.

Example 2.3.8.6. If $A = (4,8) : (1,4)$ and $B = (2,4) : (2,4)$, then

$$A \oslash B = ((2,4),(2,2)) : ((2,4),(1,16)).$$

Example 2.3.8.7. If $A = (4,6) : (1,40)$ and $B = 6 : 4$, then

$$A \oslash B = (6,4) : (40,1).$$

Example 2.3.8.8. If $A = (4,6,2,4,2,5) : (36,1,18,0,0,144)$ and $B = (4,10) : (1,192)$, then

$$A \oslash B = (((4,(2,5)),(6,2,4)) : ((36,(0,144)),(1,18,0))$$

Example 2.3.8.9. If $A = (8,(4,4))$ and $B = (2,(8,16))$, then

$$A \oslash B = ((2,2),(2,(4,4))) : ((4,8),(2,(8,16))).$$

2.3.9   Logical product

In this section, we define the **logical product** of layouts.

Definition 2.3.9.1. Suppose A and B are layouts, and suppose

$$A^c = \mathsf{comp}\big(A, \mathsf{size}(A) \cdot \mathsf{cosize}(B)\big)$$

is the complement of A with respect to $\mathsf{size}(A) \cdot \mathsf{cosize}(B)$. Then the *logical product* of A and B is the layout

$$A \otimes B = (A, A^c \circ B).$$

Observation 2.3.9.2. By Proposition 2.3.7.10 and Proposition 2.3.7.11, if we let

$$\widetilde{A^c} = \mathsf{comp}(A, N)$$

for any valid $N \geq \mathsf{size}(A) \cdot \mathsf{cosize}(B)$, then

$$A^c \circ B = \tilde{A}^c \circ B.$$

This means that when computing $A \otimes B$, we can take A^c to be any sufficiently large (sorted) complement of A.

Example 2.3.9.3. If $A = (2,2) : (5,10)$ and $B = (3,5) : (5,1)$ are the layouts

then $A \otimes B$ is the layout
$$A \otimes B = ((2,2),(3,5)) : ((5,10),(20,1))$$
as depicted below.

0	20	40	1	21	41	2	22	42	3	23	43	4	24	44
5	25	45	6	26	46	7	27	47	8	28	48	9	29	49
10	30	50	11	31	51	12	32	52	13	33	53	14	34	54
15	35	55	16	36	56	17	37	57	18	38	58	19	39	59

Example 2.3.9.4. If $A = (3,3) : (6,1)$ and $B = (10,12) : (24,2)$, then
$$A \otimes B = \big((3,3),(10,12)\big) : \big((6,1),(216,18)\big).$$

Example 2.3.9.5. If $A = (2,10) : (1680,4)$ and $B = (4,9) : (2,56)$, then
$$A \otimes B = ((2,10),((2,2),(3,3))) : ((1680,4),((2,40),(560,3360))).$$

Example 2.3.9.6. If $A = (4,(2,2)) : (9,(1,3))$ and $B = ((2,4),8) : ((1,4),2)$, then
$$A \otimes B = ((4,(2,2)),((2,4),8)) : ((9,(1,3)),((36,144),72)).$$

2.3.10 Tractable layouts

In this section we define an especially well-behaved class of layouts, called *tractable* layouts. We will see that tractable layouts are precisely the layouts which arise from a certain category **Nest**.

Definition 2.3.10.1. We say a layout L is *tractable* if the flat layout $L^\flat$ is tractable, in the sense of Definition 2.1.8.1. Explicitly, L is tractable if the flat layout
$$\mathsf{sort}(L^\flat) = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$
is such that for each $1 \le i < m$, we have

1. $d_i = 0$, or

2. $s_i d_i$ divides d_{i+1}.

Example 2.3.10.2. The layout
$$L = (((12))) : (((17)))$$
is tractable. More generally, any layout L of length 1 is tractable.

Example 2.3.10.3. The layout
$$L = ((2,4),32) : ((1,2),8)$$
is tractable. More generally, any column-major layout is tractable.

Example 2.3.10.4. The layout

$$L = (2, (4, 32)) : (128, (32, 1))$$

is tractable. More generally, any row-major layout L is tractable.

Example 2.3.10.5. The layout

$$L = ((3, 3), (1, 3), (3, 1, 3)) : ((81, 1), (0, 8), (3, 0, 27))$$

is tractable. More generally, any compact layout is tractable.

Example 2.3.10.6. The layout
$$L = ((3, 7, 7)) : ((0, 15, 0))$$
is tractable. More generally, any layout with exactly one non-zero stride entry is tractable.

Example 2.3.10.7. The layout

$$L = (2, (2, (2, 2))) : (1, (2048, (16, 64)))$$

is tractable. More generally, any complementable layout is tractable.

Example 2.3.10.8. The layout

$$L = ((8, 8), (5, 5)) : ((8, 1), (10, 2))$$

is not tractable. In particular, this shows that the concatenation (L_1, L_2) of tractable layouts L_1 and L_2 need not be tractable.

Chapter 3

Categories of layouts

Having thoroughly explored the algebra of layouts, we now turn our attention to the mathematical heart of this work: realizing layouts as morphisms in suitably-defined categories. Along the way, we develop a *graphical calculus of layout diagrams* that affords more straightforward computation of layout operations.

3.1 The category **Tuple**

In this section, we define a category **Tuple** whose objects are tuples of positive integers, and whose morphisms we call *tuple morphisms*. Each tuple morphism $f : S \to T$ encodes a flat layout L_f. Composition of tuple morphisms is compatible with layout composition, in that if f and g are composable tuple morphisms, then

$$L_{g \circ f} = L_g \circ L_f.$$

We define a realization functor (Construction 3.1.4.4)

$$|\cdot| : \textbf{Tuple} \to \textbf{FinSet}$$

which recovers the layout function of L_f via the formula

$$|f| = \Phi_{L_f}^{\mathsf{size}(T)}.$$

We develop an "algebra of tuple morphisms" which includes operations such as *sort* (Section 3.1.5.3), *coalesce* (Section 3.1.5.4), *complement* (Section 3.1.5.6), *concatenate* (Section 3.1.5.5), *flat division* (Section 3.1.5.7), and *flat products* (Section 3.1.5.8), which are compatible with the corresponding operations on flat layouts.

3.1.1 Basic definitions

Definition 3.1.1.1. Let $\textbf{Fin}_*$ denote the category whose objects are the pointed finite sets

$$\langle m \rangle_* = \{*, 1, 2, \ldots, m\}$$

for $m \geq 0$, and whose morphisms $\alpha : \langle m \rangle_* \to \langle n \rangle_*$ are functions satisfying $\alpha(*) = *$. We call these morphisms *pointed maps*, or simply *maps*.

Aside 3.1.1.2. $\textbf{Fin}_*$ is a *skeleton* of the category $\textbf{FinSet}_*$ of finite pointed sets.

Notation 3.1.1.3. If the codomain of a pointed map $\alpha : \langle m \rangle_* \to \langle n \rangle_*$ is understood, we sometimes write

$$\alpha = (\alpha(1), \dots, \alpha(m))$$

as a tuple of length m with entries in $\langle n \rangle_*$.

Example 3.1.1.4. There is a morphism $\alpha : \langle 4 \rangle_* \to \langle 6 \rangle_*$ in **Fin**$_*$ given by

$$\alpha = (2, 1, *, 6),$$

which we can visualize using the following diagram.

$$\alpha$$
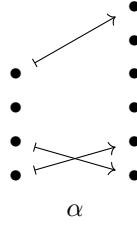
Note that the bullet corresponding to entry 3 does not support an arrow, reflecting the fact that it gets sent to $*$.

Example 3.1.1.5. There is a morphism $\beta : \langle 5 \rangle_* \to \langle 3 \rangle_*$ in **Fin**$_*$ given by

$$\beta = (*, 1, 2, 3, *) ,$$

which we can visualize using the following diagram.

$$\beta$$
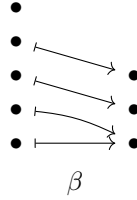
Example 3.1.1.6. For any $m \geq 0$, there is a unique morphism in **Fin**$_*$ of the form $\pi : \langle m \rangle_* \to \langle 0 \rangle_*$, namely

$$\pi = (*, \dots, *).$$

Example 3.1.1.7. For any $n \geq 0$, there is a unique morphism in **Fin**$_*$ of the form $\delta : \langle 0 \rangle_* \to \langle m \rangle_*$, namely

$$\delta = ().$$

Aside 3.1.1.8. The category **Fin**$_*$ is the category of operators for the commutative operad, so we sometimes write

$$\textbf{Fin}_* = \textbf{Comm}^\otimes.$$

We are especially interested in *tractable* morphisms in **Fin**$_*$, which we define below.

Definition 3.1.1.9. We say a pointed map $\alpha : \langle m \rangle_* \to \langle n \rangle_*$ is *tractable* if for any $j \in \langle n \rangle \subset \langle n \rangle_*$, the preimage $\alpha^{-1}(j)$ is empty or consists of a single element.

Example 3.1.1.10. The maps
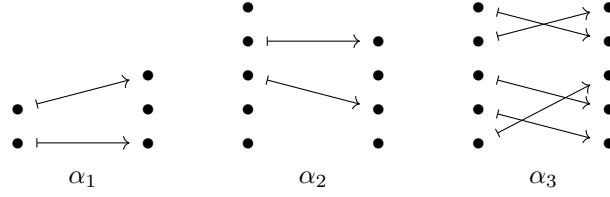
are tractable, while the maps
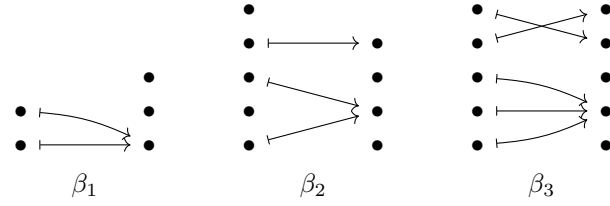
are not tractable

Remark 3.1.1.11. If we represent a morphism $\alpha : \langle m \rangle_* \to \langle n \rangle_*$ in **Fin**$_*$ as a tuple, i.e.

$$\alpha = (\alpha(1), \ldots, \alpha(m))$$

then α is tractable if and only if no positive integer occurs more than once in α.

Aside 3.1.1.12. The wide subcategory

$$\mathbf{E}_0^{\otimes} \subset \mathbf{Comm}^{\otimes} = \mathbf{Fin}_*$$

on the tractable pointed maps is the category of operators for the $\mathbf{E}_0$ operad.

Definition 3.1.1.13. Let **Tuple** denote the category whose objects are tuples

$$S = (s_1, \ldots, s_m)$$

of positive integers, where a morphism

$$f : (s_1, \ldots, s_m) \to (t_1, \ldots, t_n)$$

is specified by a tractable pointed map $\alpha : \langle m \rangle_* \to \langle n \rangle_*$ satisfying the property that

- if $1 \le i \le m$ and $\alpha(i) \ne *$, then $s_i = t_{\alpha(i)}$.

We say that such a morphism f *lies over* α, and refer to f as a *tuple morphism*.

Notation 3.1.1.14. If $f : (s_1, \ldots, s_m) \to (t_1, \ldots, t_n)$ is a tuple morphism which lies over α, then we sometimes depict f as

$$(s_1, \ldots, s_m) \xrightarrow[\alpha]{f} (t_1, \ldots, t_n).$$

The graphical calculus of layouts we develop is based on the natural visualizations of morphisms in **Tuple**, as exemplified below.

88

Example 3.1.1.15. The tuple morphism

$$(3, 128, 128) \xrightarrow[(1,3,5)]{f} (3, 2, 128, 2, 128)$$

can be visualized using the following diagram.

$$
\begin{array}{ccc}
& & 128 \\
& & 2 \\
128 & \nearrow & 128 \\
128 & \nearrow & 2 \\
3 & \longmapsto & 3 \\
& f &
\end{array}
$$
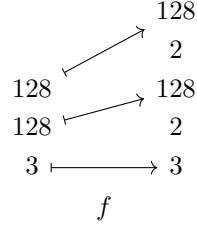
Example 3.1.1.16. The tuple morphism

$$(3, 128, 128) \xrightarrow[(*,2,1)]{g} (128, 128)$$

can be visualized using the following diagram.

$$
\begin{array}{ccc}
128 & & \\
128 & \longmapsto & 128 \\
3 & & 128 \\
& g &
\end{array}
$$
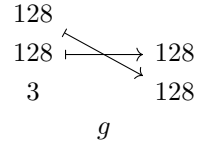
Example 3.1.1.17. The tuple morphism

$$(16, 16, 16, 1, 32) \xrightarrow[(*,*,1,*,2)]{h} (16, 32, 1, 1)$$

can be visualized using the following diagram.

$$
\begin{array}{ccc}
32 & & \\
1 & & 1 \\
16 & & 1 \\
16 & & 32 \\
16 & & 16 \\
& h &
\end{array}
$$
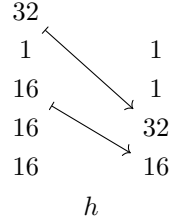
Observation 3.1.1.18. We can relate the category **Tuple** to some well-known operads as follows. Let $\mathbb{Z}_{>0}^{\mathsf{div}}$ denote the poset of positive integers under the divisibility relation, considered as a symmetric monoidal category with product given by multiplication of integers. Let $(\mathbb{Z}_{>0}^{\mathsf{div}})^{\otimes}$ denote the category of operators of $\mathbb{Z}_{>0}^{\mathsf{div}}$. Then there are evident functors

$$\textbf{Tuple} \to (\mathbb{Z}_{>0}^{\mathsf{div}})^{\otimes},$$

and

$$\textbf{Tuple} \to \mathbf{E}_0^{\otimes},$$

such that the diagram

$$
\begin{array}{ccc}
\textbf{Tuple} & \longrightarrow & (\mathbb{Z}_{>0}^{\mathsf{div}})^{\otimes} \\
\downarrow & & \downarrow \\
\mathbf{E}_0^{\otimes} & \longrightarrow & \textbf{Comm}^{\otimes}
\end{array}
$$

89

commutes. This exhibits **Tuple** as the wide subcategory of the pullback operad

$$\mathbf{Tuple} \subset \mathsf{E}_0^\otimes \times_{\mathbf{Comm}^\otimes} (\mathbb{Z}_{>0}^{\mathsf{div}})^\otimes$$

on the morphisms

$$(s_1, \ldots, s_m) \xrightarrow[\alpha]{f} (t_1, \ldots, t_n)$$

satisfying

$$\alpha(i) \neq 1 \quad \Rightarrow \quad s_i = t_{\alpha(i)}.$$

3.1.2 From tuple morphisms to flat layouts

The impetus for working with the category **Tuple** is that each tuple morphism f encodes a flat layout L_f. Moreover, each tractable layout L gives rise to a tuple morphism f_L. We prove as Proposition 3.1.2.10 that these constructions are in some sense inverses, and that tractable layouts are precisely those encoded by tuple morphisms.

Construction 3.1.2.1. Suppose

$$(s_1, \ldots, s_m) \xrightarrow[\alpha]{f} (t_1, \ldots, t_n)$$

is a tuple morphism. We define L_f to be the flat layout whose shape

$$\mathsf{shape}(L_f) = (s_1, \ldots, s_m)$$

is the domain of f, and whose stride

$$\mathsf{stride}(L_f) = (d_1, \ldots, d_m)$$

is defined by the formula

$$d_i = \begin{cases} 0 & \alpha(i) = * \\ \prod_{j < \alpha(i)} t_j & \alpha(i) \neq *. \end{cases}$$

We refer to L_f as the *layout encoded by f* or the *layout associated to f*.

Example 3.1.2.2. The tuple morphism

of Example 3.1.1.15 encodes the layout

$$L_f = (3, 128, 128) : (1, 6, 1536).$$

Note that computing the stride via the formula in Construction 3.1.2.1 amounts to following the arrow from a specific shape entry to its target entry and multiplying together all entries below that one (taking the empty product to equal 1).

Example 3.1.2.3. The tuple morphism

$$(3, 128, 128) \xrightarrow[(*,2,1)]{g} (128, 128)$$

of Example 3.1.1.16 encodes the layout

$$L_g = (3, 128, 128) : (0, 128, 1).$$

Example 3.1.2.4. The tuple morphism

$$(16, 16, 16, 1, 32) \xrightarrow[(*,*,1,*,2)]{h} (16, 32, 1, 1)$$

of Example 3.1.1.17 encodes the layout

$$L_h = (16, 16, 16, 1, 32) : (0, 0, 1, 0, 16).$$

We have seen how to compute the flat layout L_f encoded by a tuple morphism f. On the other hand, if L is *tractable*, then we can go in the other direction, constructing a tuple morphism f which encodes L. Recall from Definition 2.1.8.1 that a flat layout L is *tractable* if

$$\mathsf{sort}(L) = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

satisfies the following property:

$$\text{If } 1 \leq i < m, \text{ then } d_i = 0, \text{ or } s_i d_i \text{ divides } d_{i+1}.$$

Construction 3.1.2.5. Suppose $L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$ is tractable, and set

$$\mathsf{sort}(L) = (s'_1, \ldots, s'_m) : (d'_1, \ldots, d'_m),$$

so there is some permutation $\sigma \in \Sigma_m$ such that $\mathsf{sort}(L) = L^\sigma$. In other words, $s'_i = s_{\sigma(i)}$ and $d'_i = d_{\sigma(i)}$ for each $1 \leq i \leq m$. If each d'_i is nonzero, then let $k = 0$. Otherwise, let k be the largest integer such that $d'_k = 0$. Let $\ell = 2(m - k)$, and let

$$(t'_1, \ldots, t'_\ell) = \left(d'_{k+1}, s'_{k+1}, \frac{d'_{k+2}}{s'_{k+1} d'_{k+1}}, s'_{k+2}, \frac{d'_{k+3}}{s'_{k+2} d'_{k+2}}, \ldots, \frac{d'_m}{s'_{m-1} d'_{m-1}}, s'_m \right).$$

We define

$$f'_L : (s_1, \ldots, s_m) \to (t'_1, \ldots, t'_\ell)$$

to be the tuple morphism lying over the map $\alpha : \langle m \rangle_* \to \langle \ell \rangle_*$ given by

$$\alpha'(i) = \begin{cases} * & \sigma^{-1}(i) \leq k \\ 2(\sigma^{-1}(i) - k) & k + 1 \leq \sigma^{-1}(i) \leq m. \end{cases}$$

Let $J = \{j_1 < \cdots < j_n\} \subset \langle \ell \rangle$ denote the collection of indices such that j_i is even or $t_{j_i} \neq 1$. Let

$$(t_1, \ldots, t_n) = (t'_{j_1}, \ldots, t'_{j_n}),$$

and let $\iota : \langle n \rangle_* \to \langle \ell \rangle_*$ be the inclusion map $i \mapsto j_i$. Then by construction, the map α' factors as $\alpha' = \iota \circ \alpha$, and we define the *standard representation* of L to be the tuple morphism

$$(s_1, \ldots, s_m) \xrightarrow[\alpha]{f_L} (t_1, \ldots, t_n).$$

Example 3.1.2.6. If
$$L = (2, 2) : (3, 30),$$
then L is tractable, and the standard representation of L is the tuple morphism

$$
\begin{array}{ccc}
& & 2 \\
& \nearrow & 5 \\
2 & \rightarrow & 2 \\
2 & \rightarrow & 3 \\
& f_L &
\end{array}
$$
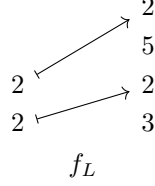
Note that, informally, computing f_L via Construction 3.1.2.5 amounts to

- initializing the codomain as (),

- traversing the non-zero strides of L in increasing order,

- if d_j is the current stride, and d_i is the previously visited stride, appending

  - (s_j) if $s_i d_i = d_j$, or
  - $\left(\frac{d_j}{s_i d_i}, s_j\right)$ if $s_i d_i < d_j$,

  and

- mapping $s_j \mapsto s_j$.

Example 3.1.2.7. If
$$L = (128, 128) : (128, 1),$$
then L is tractable, and the standard representation of L is the tuple morphism

$$
\begin{array}{ccc}
128 & \searrow \nearrow & 128 \\
128 & \nearrow \searrow & 128 \\
& f_L &
\end{array}
$$
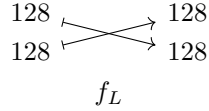
Example 3.1.2.8. If
$$L = (2, 2, 2, 2) : (24, 0, 3, 480),$$
then L is tractable, and the standard representation of L is the tuple morphism

$$
\begin{array}{ccc}
& & 2 \\
& \nearrow & 10 \\
2 & \rightarrow & 2 \\
2 & \searrow \nearrow & 4 \\
2 & \nearrow \searrow & 2 \\
2 & \rightarrow & 3 \\
& f_L &
\end{array}
$$
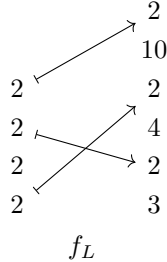
Let's justify that the tuple morphism f_L of Construction 3.1.2.5 does, in fact, encode the layout L.

Lemma 3.1.2.9. *Suppose L is a tractable flat layout, and $f = f_L$ is the standard representation of L. Then the layout encoded by f is*
$$L_f = L.$$

Proof. Suppose $L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$ is tractable, and let

$$(s_1, \ldots, s_m) \xrightarrow[\alpha]{f} (t_1, \ldots, t_n)$$

be the standard representation of L. Clearly

$$\mathsf{shape}(L_f) = (s_1, \ldots, s_m) = \mathsf{shape}(L).$$

We need to check that $\mathsf{stride}(L_f) = \mathsf{stride}(L)$. In other words, we need to check that for any $1 \leq i \leq m$, we have

$$d_i = \begin{cases} 0 & \alpha(i) = * \\ \prod_{j < \alpha(i)} t_j & \alpha(i) \neq *. \end{cases}$$

We borrow the notation of Construction 3.1.2.5. If $\alpha(i) = *$, then $\alpha'(i) = *$, and so $\sigma^{-1}(i) \leq k$. This implies

$$d_i = d'_{\sigma^{-1}(i)} = 0.$$

Suppose otherwise that $\alpha(i) \neq *$. Then $\alpha'(i) \neq *$, and so $k + 1 \leq \sigma^{-1}(i) \leq m$. We compute

$$\prod_{\substack{j < \alpha(i)}} t_j = \prod_{\substack{j' < \alpha'(i) \\ t'_{j'} \neq 1}} t'_{j'} = \prod_{\substack{j' < \alpha'(i)}} t'_{j'} = \prod_{\substack{j' < 2(\sigma^{-1}(i) - k)}} t'_{j'} = d'_{k+1} \cdot \left(\prod_{v=1}^{\sigma^{-1}(i) - (k+1)} s'_{k+v} \frac{d'_{k+v+1}}{s'_{k+v} d'_{k+v}} \right)$$

$$= d'_{\sigma^{-1}(i)}$$

$$= d_i.$$

$\square$

We have proved that if L is a tractable flat layout, then there exists a tuple morphism f which encodes L. Next, we prove the converse, which implies that tractable flat layouts are precisely the layouts encoded by tuple morphisms.

Proposition 3.1.2.10. *Suppose L is a flat layout. Then there exists a tuple morphism f encoding L if and only if L is tractable.*

Proof. First, suppose L is a flat layout, and $f : (s_1, \ldots, s_m) \to (t_1, \ldots, t_n)$ is a tuple morphism with $L_f = L$. We want to show that L_f is tractable. Let

$$\mathsf{sort}(L) = (s'_1, \ldots, s'_m) : (d_1, \ldots, d_m)$$

be the sorting of L, and suppose that $1 \leq i < m$. We will argue that $d_i = 0$, or $s'_i d_i$ divides d_{i+1}. If $d_i = 0$, then we are done. Suppose otherwise that $d_i \neq 0$. Then

$$d_i = \prod_{j < k} t_j$$

for some $1 \leq k \leq n$ with $s'_i = t_k$. Since $d_{i+1} \geq d_i$, we know that $d_{i+1} \neq 0$, so d_{i+1} has the form

$$d_{i+1} = \prod_{j < \ell} t_j$$

for some $1 \leq \ell \leq n$. There are two cases to consider:

- (Case 1) If $\ell > k$, then

$$d_{i+1} = \prod_{j<\ell} t_j = \left(\prod_{j\leq k} t_j\right)\left(\prod_{k<j<\ell} t_j\right) = s_i' d_i \left(\prod_{k<j<\ell} t_j\right),$$

so $s_i' d_i$ divides d_{i+1}.

- (Case 2) If $\ell \leq k$, then since

$$\prod_{j<\ell} t_j = d_{i+1} \geq d_i = \prod_{j<k} t_j,$$

we must have

$$t_\ell = \cdots = t_{k-1} = 1,$$

and

$$d_{i+1} = d_i.$$

In particular, we have $s_{i+1}' = t_\ell = 1$. But since $\mathsf{sort}(L_f)$ is sorted and $d_{i+1} = d_i$, we have $s_i' \leq s_{i+1}' = 1$, so $s_i' = 1$. We deduce that

$$s_i' d_i = d_{i+1},$$

so in particular, $s_i' d_i$ divides d_{i+1}.

We conclude that L is tractable.

Next, suppose that L is tractable. Then we can take $f = f_L$ to be the standard representation of L (see Construction 3.1.2.5), in which case, by Lemma 3.1.2.9, we have $L = L_f$. $\square$

Remark 3.1.2.11. It is important to note that there are many different tuple morphisms which give rise to the same layout. For example, each of the tuple morphisms shown below
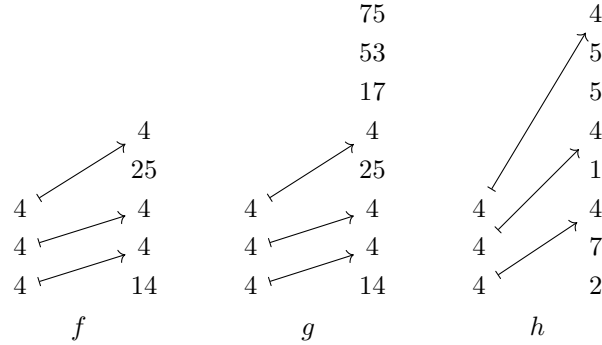
encodes the layout

$$L_f = L_g = L_h = (4,4,4) : (14, 56, 5600).$$

Among these, f is the simplest: There are no extraneous entries lying above the image of f (unlike g), and the entries not hit by f are condensed (unlike h). To make precise the simplicity of f among these morphisms, we introduce the notion of *standard form*.

Definition 3.1.2.12. Suppose

$$(s_1,\ldots,s_m) \xrightarrow[\alpha]{f} (t_1,\ldots,t_n)$$

is a tuple morphism. We say f has *standard form* if the following conditions hold:

1. If $n > 1$, then $n \in \mathsf{Image}(\alpha)$.

2. If $1 \le j < n$, then

$$j \notin \mathsf{Image}(\alpha) \quad \Rightarrow \quad \begin{array}{l} t_j \ne 1, \text{ and} \\ j+1 \in \mathsf{Image}(\alpha) \end{array}$$

Example 3.1.2.13. The tuple morphisms f of Remark 3.1.2.11 has standard form, while g and h do not.

Example 3.1.2.14. The tuple morphisms
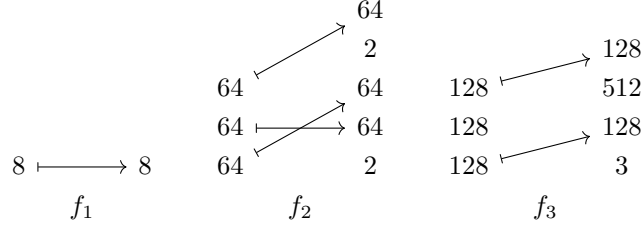
have standard form, while the tuple morphisms
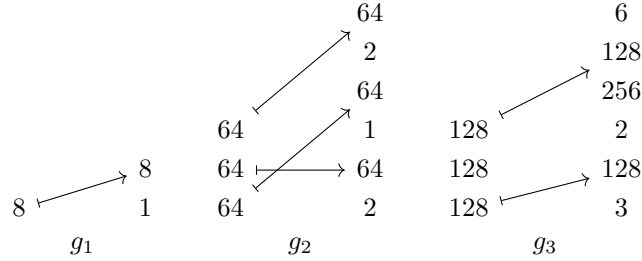
do not.

Example 3.1.2.15. If L is a tractable layout, then by construction, the standard representation f_L of L has standard form.

If we restrict to tuple morphisms of standard form, then there is almost a one-to-one correspondence with tractable layouts. However, there is one problematic case we need to exclude, as explicated in the following example.

Example 3.1.2.16. Consider the tuple morphisms f and g shown below.
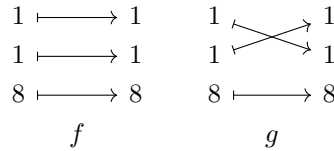
Both f and g have standard form, and

$$L_f = (8, 1, 1) : (1, 8, 8) = L_g.$$

This example illustrates that the presence of entries of the form $s_i = 1$ and $\alpha(i) \ne *$ can lead to non-uniqueness of a representing tuple morphism of standard form. On the layout side, this corresponds to shape entries $s_i = 1$ with stride $d_i \ne 0$. In order to exclude such pathological examples, we introduce the notion of *non-degeneracy*.

Definition 3.1.2.17. Suppose

$$(s_1, \ldots, s_m) \xrightarrow[\alpha]{f} (t_1, \ldots, t_n)$$

is a tuple morphism and

$$L = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

is a flat layout.

1. We say f is *non-degenerate* if
$$s_i = 1 \quad \Rightarrow \quad \alpha(i) = *.$$

2. We say L is *non-degenerate* if
$$s_i = 1 \quad \Rightarrow \quad d_i = 0.$$

Observation 3.1.2.18. If f is a non-degenerate tuple morphism, then the layout L_f encoded by f is non-degenerate. Conversely, if L is a non-degenerate flat layout, then the standard representation f_L of L is non-degenerate.

Observation 3.1.2.19. Restricting to non-degenerate flat layouts is no real loss of generality. If L is an arbitrary flat layout, then $\mathsf{filter}(L)$ is a non-degenerate flat layout with the same coordinate function and layout function as L.

The essential property of non-degenerate tuple morphisms of standard form is that they are characterized by the layouts which they encode. This is made precise as follows.

Lemma 3.1.2.20. *Suppose f and g are non-degenerate tuple morphisms of standard form. If $L_f = L_g$, then $f = g$.*

Proof. Suppose

$$(s_1, \ldots, s_m) \xrightarrow[\alpha]{f} (t_1, \ldots, t_n)$$

and

$$(s_1, \ldots, s_m) \xrightarrow[\beta]{g} (u_1, \ldots, u_p)$$

are non-degenerate tuple morphisms of standard form with

$$L_f = (s_1, \ldots, s_m) : (d_1, \ldots, d_m) = L_g.$$

We want to show that $f = g$. First, we will argue that $(t_1, \ldots, t_n) = (u_1, \ldots, u_p)$. Let

$$X = \{t_1 \cdots t_j \mid 1 \le j \le n\}$$
$$Y = \{u_1 \cdots u_k \mid 1 \le k \le p\}$$

denote the sets of prefix products of $(t_1, \ldots, t_n)$ and $(u_1, \ldots, u_p)$, respectively. We claim $X = Y$, since each of these sets is equal to

$$Z = \{d_i, s_i d_i \mid 1 \le i \le m \text{ and } d_i \ne 0\}.$$

Lets argue that $X = Z$. Suppose $1 \le j \le n$. If there exists some $i \in \langle m \rangle$ with $\alpha(i) = j$, then $t_1 \cdots t_j = s_i d_i$. On the other hand, if j is not in the image of α, then since f has standard form, there exists some $i \in \langle m \rangle$ such that $\alpha(i) = j + 1$, in which case $t_1 \cdots t_j = d_i$. This proves that $X \subseteq Z$.

Conversely, if $1 \leq i \leq m$ and $d_i \neq 0$, then $d_i = t_1 \cdots t_{\alpha(i)-1}$ and $s_i d_i = t_1 \cdots t_{\alpha(i)}$, which proves $Z \subseteq X$. We deduce that $X = Z$. The same argument proves $Y = Z$.

Since f and g are non-degenerate of standard form, we know that each t_j and each u_k is greater than 1, which implies

$$t_1 < t_1 t_2 < \cdots < t_1 \cdots t_n,$$
$$u_1 < u_1 u_2 < \cdots < u_1 \cdots u_p,$$

and since $X = Y$, it follows that $n = p$, and $t_1 \cdots t_j = u_1 \cdots u_j$ for each $1 \leq j \leq n$. We deduce that $(t_1, \ldots, t_n) = (u_1, \ldots, u_p)$.

Next, we need to argue that $\alpha = \beta$. Suppose for contradiction that there exists some $i \in \langle m \rangle$ with $\alpha(i) \neq \beta(i)$. There are two cases to consider.

- If $\alpha(i) = * \neq \beta(i)$, then

$$0 = d_i = t_1 \cdots t_{\beta(i)-1},$$

  a contradiction. The case $\alpha(i) \neq * = \beta(i)$ is analogous.

- If $\alpha(i) \neq * \neq \beta(i)$, then without loss of generality we may assume $\alpha(i) < \beta(j)$, in which case

$$d_i = t_1 \cdots t_{\alpha(i)-1} < t_1 \cdots t_{\beta(i)-1} = d_i,$$

  a contradiction.

We deduce that $\alpha = \beta$, so $f = g$. $\qquad\square$

We are now ready to prove our correspondence theorem, which identifies non-degenerate tuple morphisms of standard form with non-degenerate tractable flat layouts.

Theorem 3.1.2.21. *The maps*

$$f \longmapsto L_f$$

$$\left\{ \begin{array}{c} \textit{Non-degenerate} \\ \textit{tuple morphisms of} \\ \textit{standard form} \end{array} \right\} \longleftrightarrow \left\{ \begin{array}{c} \textit{Non-degenerate} \\ \textit{tractable flat} \\ \textit{layouts} \end{array} \right\}$$

$$f_L \longleftarrow L$$

of Constructions 3.1.2.1 and 3.1.2.5 determine a one-to-one correspondence between non-degenerate tuple morphisms of standard form, and non-degenerate tractable flat layouts.

Proof. We want to show that the constructions $f \mapsto L_f$ and $L \mapsto f_L$ are inverses, when restricted to tuple morphisms and layouts of the stated form. If L is a non-degenerate tractable flat layout, then by Lemma 3.1.2.9 we have $L_{f_L} = L$. Suppose next that f is a non-degenerate tuple morphism of standard form and $L = L_f$ is the layout encoded by f. Since f and f_{L_f} are non-degenerate tuple morphisms of standard form, and the layouts encoded by these tuple morphsims are equal, it follows from Lemma 3.1.2.20 that $f = f_{L_f}$. $\qquad\square$

3.1.3 Examples

In this section, we introduce some important families of tuple morphisms, and describe the flat layouts to which they give rise.

Example 3.1.3.1 (Identity morphisms)**.** We say a tuple morphism f is an *identity morphism* if $f = \mathsf{id}_S$ for some tuple S. If $f = \mathsf{id}_S$ is an identity morphism, then L_f is the column-major layout with shape S. For instance, here is an example of an identity morphism f together with its associated layout L_f.

$$
\begin{array}{c}
4 \longmapsto 4 \\
4 \longmapsto 4 \\
2 \longmapsto 2 \\
2 \longmapsto 2 \\
2 \longmapsto 2 \\
f
\end{array}
\qquad \rightsquigarrow \qquad L_f = (2,2,2,4,4) : (1,2,4,8,32)
$$

Example 3.1.3.2 (Isomorphisms)**.** A tuple morphism $f : S \to T$ is an *isomorphism* if there is a tuple morphism $g : T \to S$ such that $g \circ f = \mathsf{id}_S$ and $f \circ g = \mathsf{id}_T$. If f is an isomorphism, then its associated layout L_f is compact. For instance, here is an isomorphism f together with its associated layout L_f.

$$
\begin{array}{c}
4 \quad 2 \\
4 \quad 4 \\
2 \quad 4 \\
2 \quad 2 \\
2 \quad 2 \\
f
\end{array}
\qquad \rightsquigarrow \qquad L_f = (2,2,2,4,4) : (2,1,64,4,16)
$$

.

Observation 3.1.3.3. Note that if a tuple morphism

$$
(s_1, \ldots, s_m) \xrightarrow[\alpha]{f} (t_1, \ldots, t_n)
$$

is an isomorphism, then $\alpha : \langle m \rangle_* \to \langle m \rangle_*$ is a bijection, and so $\alpha \mid_{\langle m \rangle} \in \Sigma_m$ is a permutation. Conversely, if $\sigma \in \Sigma_m$ is a permutation, and $(s_1, \ldots, s_m)$ is a tuple of positive integers, then we may construct the isomorphism

$$
(s_1, \ldots, s_m) \xrightarrow[\sigma_*]{f} (s_{\sigma(1)}, \ldots, s_{\sigma(m)}).
$$

We conclude that there is a one-to-one correspondence between tuple isomorphisms f with domain $(s_1, \ldots, s_m)$, and permutations in Σ_m.

Example 3.1.3.4 (Projections)**.** Suppose $S = (s_1, \ldots, s_m)$ is a shape, and suppose

$$
\{i_1 < \cdots < i_r\} \subset \langle m \rangle
$$

is some subset. Let

$$
(s_1, \ldots, s_m) \xrightarrow[\alpha]{p} (s_{i_1}, \ldots, s_{i_r})
$$

be the tuple morphism lying over the map α with

$$\alpha(x) = \begin{cases} j & x = i_j \\ * & \text{else.} \end{cases}$$

We call p the *projection of* $(s_1, \ldots, s_m)$ *onto* $(s_{i_1}, \ldots, s_{i_r})$. The layout encoded by p is

$$L_p = (s_1, \ldots, s_m) : (d_1, \ldots, d_m),$$

where

$$d_i = \begin{cases} s_{i_1} \cdots s_{i_{j-1}} & i = i_j \text{ for some } 1 \le j \le r \\ 0 & \text{otherwise.} \end{cases}$$

For instance, here is a projection p of $(64, 64, 3, 8)$ onto $(64, 3)$, together with its associated layout.

$$
\begin{array}{c}
8 \\
3 \quad \searrow \\
64 \quad \searrow \; 3 \qquad \rightsquigarrow \qquad L_p = (64, 64, 3, 8) : (1, 0, 64, 0) \\
64 \longmapsto 64 \\
p
\end{array}
$$

Example 3.1.3.5 (Dilations). Suppose $S = (s_1, \ldots, s_m)$ is a shape, and suppose $c_1, \ldots, c_m$ are positive integers. The tuple morphism

$$(s_1, \ldots, s_m) \xrightarrow[\;(*,2,*,4,\ldots,*,2m)\;]{f} (c_1, s_1, \ldots, c_m, s_m)$$

is called the *dilation of* $(s_1, \ldots, s_m)$ *by* $(c_1, \ldots, c_m)$. The layout L_f associated to this morphism is $L_f = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$, where

$$d_i = \prod_{j < i} c_j s_j.$$

For instance, here is the dilation f of $(512, 512)$ by $(2, 4)$, together with its associated layout.

$$
\begin{array}{c}
512 \\
\nearrow \; 4 \\
512 \quad \searrow \; 512 \qquad \rightsquigarrow \qquad L_f = (512, 512) : (2, 4096) \\
512 \quad \searrow \; 2 \\
f
\end{array}
$$

Example 3.1.3.6 (Expansions). Suppose $S = (s_1, \ldots, s_m)$ is a tuple of positive integers, and suppose $1 \le i \le m'$, so that $S' = (s_1, \ldots, s_{m'})$ divides S. Then the tuple morphism

$$(s_1, \ldots, s_{m'}) \xrightarrow[\;(1,2,\ldots,m')\;]{e} (s_1, \ldots, s_{m'}, \ldots, s_m)$$

is called the *expansion of* S' *to* S. The layout encoded by e is the column-major layout with shape $(s_1, \ldots, s_{m'})$. For instance, here is the expansion of $S' = (4, 4)$ to $S = (4, 4, 8, 8)$.

$$
\begin{array}{c}
8 \\
8 \\
4 \longmapsto 4 \qquad \rightsquigarrow \qquad L_e = (4, 4) : (1, 4) \\
4 \longmapsto 4 \\
e
\end{array}
$$

An important property of expansions is that if $f : S \to T$ is any tuple morphism and $e : T \to T'$ is an expansion, then

$$L_{e \circ f} = L_f.$$

In other words, post-composing f with an expansion does not change the layout encoded by f.

Example 3.1.3.7 (Restrictions). Suppose

$$(s_1, \ldots, s_m) \xrightarrow[\alpha]{f} (t_1, \ldots, t_n)$$

is a tuple morphism, and suppose

$$I = \{i_1 < \cdots < i_r\} \subset \langle m \rangle$$

is a subset of indices. Then the tuple morphism

$$(s_{i_1}, \ldots, s_{i_r}) \xrightarrow[\alpha \circ \iota]{f|_I} (t_1, \ldots, t_n)$$

is called the *restriction of f to I*. If the layout encoded by f is

$$L_f = (s_1, \ldots, s_m) : (d_1, \ldots, d_m),$$

then the layout encoded by $f \mid_I$ is

$$L_{f|_I} = (s_{i_1}, \ldots, s_{i_r}) : (d_{i_1}, \ldots, d_{i_r}).$$

For instance, here is the restriction $f \mid_I$ of a tuple morphism f, where $I = \{2, 4\}$.

$$L_f = (2, 16, 8, 4) : (0, 8, 1, 128)$$

$$L_{f|_I} = (16, 4) : (8, 128)$$

Example 3.1.3.8 (Entry inclusions). An important special case of the previous construction is as follows. If $f : (s_1, \ldots, s_m) \to (t_1, \ldots, t_m)$ is a tuple morphism and $1 \leq i \leq m$, then the *ith entry f_i of f* is

$$(s_i) \xrightarrow[(i)]{f_i} (t_1, \ldots, t_n)$$

If the layout encoded by f is

$$L_f = (s_1, \ldots, s_m) : (d_1, \ldots, d_m),$$

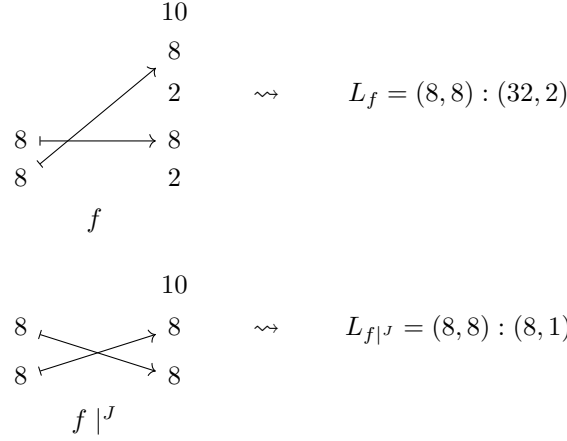then the layout encoded by f_i is

$$L_{f_i} = (s_i) : (d_i).$$

For instance, here is a tuple morphism f, and its fourth entry f_4.

$$
\begin{array}{c}
4 \\
8 \quad\quad\quad\quad 4 \\
16 \longmapsto 16 \quad\quad \rightsquigarrow \quad\quad L_f = (2, 16, 8, 4) : (0, 8, 1, 128) \\
2 \quad\quad\quad\quad 8 \\
f
\end{array}
$$

$$
\begin{array}{c}
4 \\
16 \quad\quad \rightsquigarrow \quad\quad L_{f_4} = (4) : (128) \\
4 \quad\quad 8 \\
f_4
\end{array}
$$

Remark 3.1.3.9. Given an $\langle n \rangle_* \in \mathsf{Fin}_*$, there is a morphism $\varphi_i : \langle 1 \rangle_* \to \langle n \rangle_*$ for each $i \in \langle n \rangle$ sending $* \mapsto *$ and $1 \mapsto i$. For a tuple morphism $f : (s_1, \ldots, s_m) \to (t_1, \ldots, t_n)$ lying over $\alpha : \langle m \rangle_* \to \langle n \rangle_*$, the i-th entry lies over the composite $\alpha \circ \varphi_i : \langle 1 \rangle_* \to \langle n \rangle_*$.

Example 3.1.3.10 (Factorizations). Suppose

$$
(s_1, \ldots, s_m) \xrightarrow[\alpha]{f} (t_1, \ldots, t_n)
$$

is a tuple morphism, and suppose

$$
J = \{ j_1 < \cdots < j_\ell \} \subset \langle n \rangle
$$

is a subset such that $\mathsf{Image}(\alpha) \subseteq J \cup \{*\}$. If we write $\iota : \langle \ell \rangle_* \to \langle n \rangle_*$ for the map $k \mapsto j_k$, then α factors as $\alpha = \iota \circ \bar{\alpha}$ for a unique map $\bar{\alpha} : \langle m \rangle_* \to \langle \ell \rangle_*$, and we define the *factorization of f through J* to be the tuple morphism

$$
(s_1, \ldots, s_m) \xrightarrow[\bar{\alpha}]{f|^J} (t_{j_1}, \ldots, t_{j_\ell}).
$$

If the layout encoded by f is

$$
L_f = (s_1, \ldots, s_m) : (d_1, \ldots, d_m),
$$

then the layout encoded by $f \,|^J$ is

$$
L_{f|^J} = (s_1, \ldots, s_m) : (d'_1, \ldots, d'_m),
$$

where

$$
d'_i = \frac{d_i}{\left(\displaystyle\prod_{k < \alpha(i) \text{ and } k \notin J} t_j \right)}.
$$

For instance, here is the factorization $f \mid^J$ of a tuple morphism f, where $J = \{2, 4, 5\}$.

$$
\begin{array}{c}
10 \\
8 \\
\nearrow \quad 2 \\
8 \longmapsto \longrightarrow 8 \qquad \rightsquigarrow \qquad L_f = (8,8) : (32, 2) \\
8 \qquad\quad 2 \\
f
\end{array}
$$

$$
\begin{array}{c}
10 \\
8 \longmapsto \searrow \nearrow \rightarrow 8 \qquad \rightsquigarrow \qquad L_{f\mid^J} = (8,8) : (8, 1) \\
8 \longmapsto \nearrow \searrow \rightarrow 8 \\
f \mid^J
\end{array}
$$

Remark 3.1.3.11. There is a categorical interpretation of factorizations. Borrowing the notation of Example 3.1.3.10, we may observe that there is a tuple morphism $i : (t_{j_1}, \ldots, t_{j_\ell}) \to (t_1, \ldots, t_n)$ lying over ι, and $f \mid^J$ is the pullback of f along i:

$$
\begin{array}{ccc}
(s_1, \ldots, s_m) & \xrightarrow{\;f\mid^J\;} & (t_{j_1}, \ldots, t_{j_\ell}) \\
{\scriptstyle \mathsf{id}}\downarrow & \lrcorner & \downarrow{\scriptstyle i} \\
(s_1, \ldots, s_m) & \xrightarrow[\;f\;]{} & (t_1, \ldots, t_n)
\end{array}
$$

3.1.4 Realization of tuple morphisms

As we have seen, a tuple morphism $f : S \to T$ encodes a flat layout L_f. In this section, we will construct a realization functor

$$| \cdot | : \mathsf{Tuple} \to \mathsf{FinSet}.$$

which makes this encoding explicit. The realization functor $| \cdot |$ sends a tuple morphism f to the layout function $|f|$ of L_f. In order to construct our realization functor $| \cdot |$, we first construct an auxiliary functor

$$F : \mathsf{Tuple} \to \mathsf{FinSet}$$

which we will use in our construction.

Construction 3.1.4.1. We define a functor

$$F : \mathsf{Tuple} \to \mathsf{FinSet}$$

as follows.

- For an object $S = (s_1, \ldots, s_m)$ in Tuple, we define

$$FS = [0, S) = \prod_{i=1}^{m} [0, s_i).$$

102

- For a morphism $f : (s_1, \ldots, s_m) \to (t_1, \ldots, t_n)$ in **Tuple** lying over α, we define Ff to be the map

$$[0, S) \xrightarrow{\;\;Ff\;\;} [0, T)$$

given by

$$(Ff)(x_1, \ldots, x_m) = (y_1, \ldots, y_n)$$

where

$$y_j = \begin{cases} x_i & \text{there exists } 1 \le i \le m \text{ with } \alpha(i) = j, \\ 0 & \text{else.} \end{cases}$$

One may easily verify that $F(g \circ f) = Fg \circ Ff$ and $F\mathsf{id}_S = \mathsf{id}_{FS}$, so F is in fact a functor.

Example 3.1.4.2. Suppose $f : (4, 4) \to (4, 4, 4)$ is the tuple morphism lying over $\alpha = (1, 3)$. Then

$$Ff : [0, (4, 4)) \to [0, (4, 4, 4))$$

is given by

$$(Ff)(x_1, x_2) = (x_1, 0, x_2).$$

Example 3.1.4.3. Suppose $g : (3, 256, 256, 512) \to (3, 256, 256)$ is the tuple morphism lying over $\beta = (*, 3, 2, *)$. Then

$$Fg : [0, (3, 256, 256, 512)) \to [0, (3, 256, 256))$$

is given by

$$(Fg)(x_1, x_2, x_3, x_4) = (0, x_3, x_2).$$

Construction 3.1.4.4. We define a functor

$$|\cdot| : \textbf{Tuple} \to \textbf{FinSet}$$

as follows.

- For an object $S = (s_1, \ldots, s_m)$ in **Tuple**, we define

$$|S| = [0, \mathsf{size}(S)) = \{0, 1, \ldots, \mathsf{size}(S) - 1\}.$$

- For a tuple morphism $f : S \to T$, we define

$$|f| = \mathsf{colex}_T \circ Ff \circ \mathsf{colex}_S^{-1}$$

(recall Definition 2.1.2.18).

If $f : S \to T$ and $g : T \to U$ are composable tuple morphisms then

$$\begin{aligned}
|g \circ f| &= \mathsf{colex}_U \circ F(g \circ f) \circ \mathsf{colex}_S^{-1} \\
&= \mathsf{colex}_U \circ Fg \circ Ff \circ \mathsf{colex}_S^{-1} \\
&= \mathsf{colex}_U \circ Fg \circ \mathsf{colex}_T^{-1} \circ \mathsf{colex}_T \circ Ff \circ \mathsf{colex}_S^{-1} \\
&= |g| \circ |f|
\end{aligned}$$

and if $f = \mathsf{id}_S$ is an identity morphism, then

$$
\begin{aligned}
|\mathsf{id}_S| &= \mathsf{colex}_S \circ F\mathsf{id}_S \circ \mathsf{colex}_S^{-1} \\
&= \mathsf{colex}_S \circ \mathsf{id}_S \circ \mathsf{colex}_S^{-1} \\
&= \mathsf{colex}_S \circ \mathsf{colex}_S^{-1} \\
&= \mathsf{id}_{|S|} \ ,
\end{aligned}
$$

so $|\cdot|$ does in fact specify a functor. Next, we observe that for a morphism f in **Tuple**, the map $|f|$ is the layout function of L_f. This allows us to easily deduce that composition of morphisms in **Tuple** is compatible with composition of flat layouts (see Corollary 3.1.4.6).

Lemma 3.1.4.5. *If $f : S \to T$ is a tuple morphism, then the realization $|f|$ of f is the layout function of L_f:*

$$
|f| = \Phi_{L_f}^{\mathsf{size}(T)}
$$

Proof. Let $S = (s_1, \ldots, s_m)$, $T = (t_1, \ldots, t_n)$, and let

$$
L_f = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)
$$

denote the layout associated to f, whose strides d_i are defined by the formula

$$
d_i = \begin{cases} 0 & \alpha(i) = * \\ \prod_{j < \alpha(i)} t_j & \text{else.} \end{cases}
$$

By precomposing with $\mathsf{colex}_S : \prod_{i=1}^m [0, s_i) \to [0, \mathsf{size}(S))$, it suffices to prove that for any $(x_1, \ldots, x_m) \in \prod_{i=1}^m [0, s_i)$, we have

$$
(\mathsf{colex}_T \circ Ff)(x_1, \ldots, x_m) = (x_1, \ldots, x_m) \cdot (d_1, \ldots, d_m).
$$

For a general input $(x_1, \ldots, x_m) \in \prod_{i=1}^m [0, s_i)$, we have

$$
(Ff)(x_1, \ldots, x_m) = (y_1, \ldots, y_n)
$$

where y_j is equal to x_i if $\alpha(i) = j$, and 0 otherwise. It follows that

$$
\begin{aligned}
(\mathsf{colex}_T \circ Ff)(x_1, \ldots, x_m) &= (y_1, \cdots, y_n) \cdot (1, t_1, \ldots, t_1 \cdots t_{n-1}) \\
&= \sum_{j=1}^n y_j \cdot t_1 \cdots t_{j-1} \\
&= \sum_{i=1}^m x_i d_i \\
&= (x_1, \ldots, x_m) \cdot (d_1, \ldots, d_m),
\end{aligned}
$$

as desired. $\square$

Corollary 3.1.4.6. *If f and g are non-degenerate composable tuple morphisms, then*

$$
L_{g \circ f} = L_g \circ L_f
$$

Proof. Suppose $f : S \to T$ and $g : T \to U$ are morphisms in **Tuple** lying over α and β, respectively. Write $S = (s_1, \ldots, s_m)$ and $T = (t_1, \ldots, t_n)$. We need to check that

104

1. $\mathsf{shape}(L_{g \circ f})$ *refines* $\mathsf{shape}(L_f)$: This holds since the shape of L_f and $L_{g \circ f}$ are both equal to S.

2. $L_{g \circ f}$ *is coalesced over* $\mathsf{shape}(L_f)$: This holds since the tuple morphism $g \circ f$ is non-degenerate, hence so is the layout $L_{g \circ f}$.

3. $\Phi_{L_{g \circ f}} = \Phi_{L_g} \circ \Phi_{L_f}^{\mathsf{size}(L_g)}$: Using Lemma 3.1.4.5, we have

$$\Phi_{L_{g \circ f}}^{\mathsf{size}(U)} = |g \circ f|$$
$$= |g| \circ |f|$$
$$= \Phi_{L_g}^{\mathsf{size}(U)} \circ \Phi_{L_f}^{\mathsf{size}(T)}.$$

and by postcomposing with the inclusion $[0, \mathsf{size}(U)) \subset \mathbb{Z}$, and observing that $\mathsf{size}(T) = \mathsf{size}(L_g)$, the result follows.

$\square$

3.1.5 Operations on tuple morphisms

Our next goal is to develop an "algebra of tuple morphisms", which includes operations such as *coalesce*, *complement*, *composition*, *flat division*, and *flat products*. We will prove that each of these operations is compatible with a corresponding operation on flat layouts.

3.1.5.1 Sum

The sum $f \oplus g$ of tuple morphisms f and g is obtained by concatenating the domains and codomains of f and g. In order to define this operations precisely, we first define a corresponding operation on morphisms in $\mathbf{Fin}_*$.

Definition 3.1.5.1. Suppose $\alpha : \langle m \rangle_* \to \langle n \rangle_*$ and $\beta : \langle p \rangle_* \to \langle q \rangle_*$ are morphisms in $\mathbf{Fin}_*$. We define the *sum* of α and β to be the morphism

$$\alpha \oplus \beta : \langle m + p \rangle_* \to \langle n + q \rangle_*$$

given by

$$(\alpha \oplus \beta)(x) = \begin{cases} \alpha(x) & 1 \le x \le m \\ n + \beta(x - m) & m + 1 \le x \le m + p \\ * & x = *. \end{cases}$$

This operation is associative, so we can consider the sum $\alpha_1 \oplus \cdots \oplus \alpha_k$ for any finite collection of morphisms $\alpha_1, \ldots, \alpha_k$ in $\mathbf{Fin}_*$.

Remark 3.1.5.2. If α and β are tractable pointed maps, then $\alpha \oplus \beta$ is tractable.

Now we can define the sum of morphisms in $\mathbf{Tuple}$.

Definition 3.1.5.3. Suppose $f : S \to T$ and $g : U \to V$ are tuple morphisms lying over α and β, respectively. We define the *sum* of f and g to be the tuple morphism

$$f \oplus g : S \star U \to T \star V$$

lying over $\alpha \oplus \beta$. This operation is associative, so we can consider the sum $f_1 \oplus \cdots \oplus f_k$ for any finite collection of morphisms $f_1, \ldots, f_k$ in $\mathbf{Tuple}$.

Example 3.1.5.4. Here is an example of the sum $f \oplus g$ of tuple morphisms f and g.

$$
\begin{array}{ccc}
 & & \begin{array}{ccc} & & 4 \longrightarrow 4 \\ & & 4 \longmapsto \quad 2 \end{array} \\
\begin{array}{ccc} 32 \longmapsto 32 \\ 16 \longmapsto 16 \end{array} & \begin{array}{ccc} 4 \quad\quad 4 \\ 4 \quad\quad 2 \end{array} & \begin{array}{ccc} 32 \longmapsto 32 \\ 16 \longmapsto 16 \end{array} \\
f & g & f \oplus g
\end{array}
$$
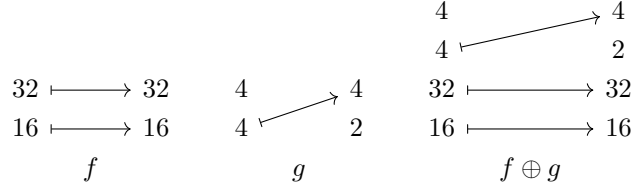
Example 3.1.5.5. Here is another example of the sum $f \oplus g$ of tuple morphisms f and g.

$$
\begin{array}{ccc}
 & & \begin{array}{ccc} 64 \longmapsto 64 \\ 512 \quad\quad 256 \\ 256 \quad\quad 512 \end{array} \\
\begin{array}{ccc} 128 \quad\quad 128 \\ 128 \quad\quad 128 \end{array} & \begin{array}{ccc} 64 \longmapsto 64 \\ 512 \quad\quad 256 \\ 256 \quad\quad 512 \end{array} & \begin{array}{ccc} 128 \quad\quad 128 \\ 128 \quad\quad 128 \end{array} \\
f & g & f \oplus g
\end{array}
$$
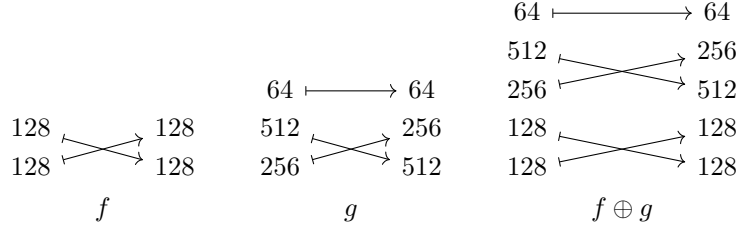
Remark 3.1.5.6. There is a categorical interpretation of the sum of tuple morphisms: if $f : S \to T$ and $g : U \to V$ are tuple morphisms, then

$$ f \oplus g : S \star U \to T \star V $$

is the coproduct of f and g in the arrow category $\mathsf{Ar}(\mathbf{Tuple})$.

3.1.5.2 Squeeze

It is often the case that we want to remove any instances of the integer 1 from our tuples. This is accomplished by the *squeeze* functor.

Definition 3.1.5.7. We define a functor

$$ \mathbf{Tuple} \xrightarrow{\text{squeeze}(-)} \mathbf{Tuple} $$

as follows. If $S = (s_1, \ldots, s_m)$ is an object in **Tuple**, we define

$$ \mathsf{squeeze}(S) = (s_{i_1}, \ldots, s_{i_k}) $$

where $\{i_1 < \cdots < i_k\} \subset \langle m \rangle$ are the indices with $s_{i_j} \neq 1$. If $f : (s_1, \ldots, s_m) \to (t_1, \ldots, t_n)$ is a tuple morphism, we define

$$ \mathsf{squeeze}(f) : \mathsf{squeeze}(S) \to \mathsf{squeeze}(T) $$

to be the tuple morphism

$$ \mathsf{squeeze}(f) = (f \mid_I) \mid^J $$

where $f \mid_I$ is the restriction of f to

$$ I = \{ i \in \langle m \rangle \mid s_i \neq 1 \} $$

as in Definition 3.1.3.7, and where $(f \mid_I) \mid^J$ is be the factorization of $f \mid_I$ through

$$ J = \{ j \in \langle n \rangle \mid t_j \neq 1 \}, $$

as in Definition 3.1.3.10.

106

Example 3.1.5.8. Here is an example of a morphism f and the corresponding morphism $\mathsf{squeeze}(f)$.
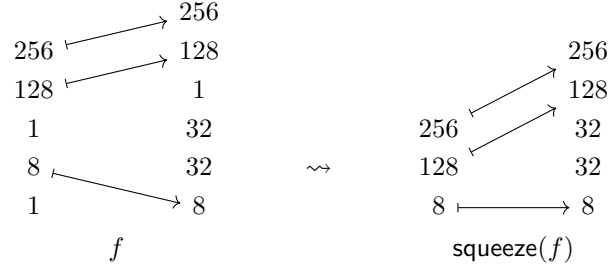
Example 3.1.5.9. If $f : (s_1, \ldots, s_m) \to (t_1, \ldots, t_n)$ is a tuple morphism, then

$$f = \mathsf{squeeze}(f) \qquad \Leftrightarrow \qquad \text{no } s_i, \, t_j \text{ is equal to } 1.$$

Proposition 3.1.5.10. *If f is a tuple morphism, then*

$$L_{\mathsf{squeeze}(f)} = \mathsf{squeeze}(L_f).$$

Proof. Suppose $f : (s_1, \ldots, s_m) \to (t_1, \ldots, t_m)$ is a tuple morphism, and let

$$L_f = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

be the flat layout associated to f. Let $I = \{i_1 < \cdots < i_{m'}\} \subset \langle m \rangle$ denote the subset of indices with $s_{i_k} \neq 1$. Then

$$
\begin{aligned}
L_{f|_I} &= (s_{i_1}, \ldots, s_{i_k}) : (d_{i_1}, \ldots, d_{i_k}) \\
&= \mathsf{squeeze}(L_f).
\end{aligned}
$$

Let $J = \{j_1 < \cdots < j_{n'}\} \subset \langle n \rangle$ denote the subset of indices with $t_{j_k} \neq 1$, so that $\mathsf{squeeze}(f) = (f \mid_I) \mid^J$. Let β denote the map over which $\mathsf{squeeze}(f)$ lies. Then

$$L_{\mathsf{squeeze}(f)} = L_{(f|_I)|^J} = (s_{i_1}, \ldots, s_{i_k}) : (d'_{i_1}, \ldots, d'_{i_k})$$

where

$$
\begin{aligned}
d'_{i_k} &= \frac{d_{i_k}}{\left(\displaystyle\prod_{\ell < \beta(k) \text{ and } \ell \notin J} t_\ell \right)} \\
&= d_{i_k}
\end{aligned}
$$

since $t_\ell = 1$ for any $\ell \notin J$. We conclude that

$$
\begin{aligned}
L_{\mathsf{squeeze}(f)} &= (s_{i_1}, \ldots, s_{i_k}) : (d_{i_1}, \ldots, d_{i_k}) \\
&= \mathsf{squeeze}(L_f).
\end{aligned}
$$

$\square$

Observation 3.1.5.11. If f is a tuple morphism, then

$$\mathsf{squeeze}(\mathsf{squeeze}(f)) = \mathsf{squeeze}(f),$$

so

$$\mathbf{Tuple} \xrightarrow{\;\mathsf{squeeze}(-)\;} \mathbf{Tuple}$$

is an idempotent functor.

3.1.5.3   Sort

The sort operation $f \mapsto \mathsf{sort}(f)$ permutes the domain of f so that the resulting morphism is *sorted*, in the following sense.

Definition 3.1.5.12. We say a tuple morphism

$$(s_1, \ldots, s_m) \xrightarrow[\alpha]{f} (t_1, \ldots, t_n)$$

is *sorted* if for any $1 \leq i, j \leq m$, the following conditions hold.

1. If $\alpha(i) = * \neq \alpha(j)$, then $i < j$.

2. If $\alpha(i) = * = \alpha(j)$, then

$$i \leq j \quad \Rightarrow \quad s_i \leq s_j.$$

3. If $\alpha(i) \neq * \neq \alpha(j)$, then

$$i \leq j \quad \Rightarrow \quad \alpha(i) \leq \alpha(j).$$

Example 3.1.5.13. The morphisms f_1, f_2, and f_3 shown below
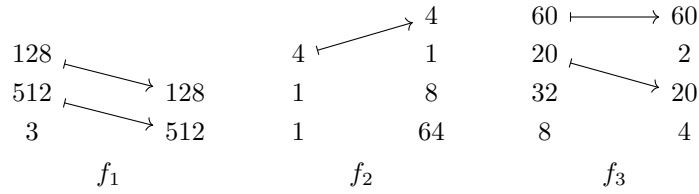
are sorted, while the morphisms g_1, g_2, and g_3 shown below
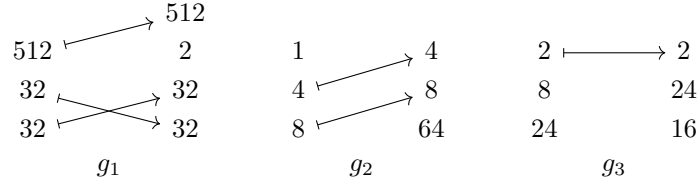
are not sorted. The morphisms g_1, g_2, and g_3 violate conditions 3, 1, and 2, respectively.

Proposition 3.1.5.14. *If f is a sorted tuple morphism, then the flat layout L_f is sorted.*

Proof. Suppose

$$(s_1, \ldots, s_m) \xrightarrow[\alpha]{f} (t_1, \ldots, t_n)$$

is sorted, and consider the layout

$$L_f = (s_1, \ldots, s_m) : (d_1, \ldots, d_m).$$

Suppose $1 \leq i < m$. We want to show that $d_i < d_{i+1}$, or $d_i = d_{i+1}$ and $s_i \leq s_{i+1}$. There are two cases to consider.

- (Case 1) Suppose that $\alpha(i) = *$, so that $d_i = 0$. If $\alpha(i+1) = *$, then $d_{i+1} = 0$ and since f is sorted we have $s_i \leq s_{i+1}$. If $\alpha(i+1) \neq *$, then $d_{i+1} \geq 1 > 0 = d_i$.

- (Case 2) Suppose that $\alpha(i) \neq *$, in which case $\alpha(i+1) \neq *$ and $\alpha(i) < \alpha(i+1)$. Then

$$d_i = \prod_{j < \alpha(i)} t_j \leq \prod_{j < \alpha(i+1)} = d_{i+1},$$

  where equality holds only if $s_i = t_{\alpha(i)} = 1$, which implies $s_i \leq s_{i+1}$.

We conclude that L_f is sorted. $\qquad\square$

Next, we define our $\mathsf{sort}(-)$ operation on **Tuple**. If f is a tuple morphism, then $\mathsf{sort}(f)$ will be obtained by precomposing f with an appropriate permutation g.

Construction 3.1.5.15. Suppose

$$(s_1, \ldots, s_m) \xrightarrow[\alpha]{f} (t_1, \ldots, t_n)$$

is a tuple morphism. We define a permutation $\sigma \in \Sigma_m$ as follows. Set

$$P = \{i \in \langle m \rangle \mid \alpha(i) = *\},$$
$$Q = \{i \in \langle m \rangle \mid \alpha(i) \neq *\},$$

so $\langle m \rangle$ is the disjoint union of P and Q. We define a linear ordering of P by $i_1 \preceq_P i_2$ if

1. $s_{i_1} < s_{i_2}$, or

2. $s_{i_1} = s_{i_2}$ and $i_1 \leq i_2$.

We define a linear ordering on Q by $j_1 \preceq_Q j_2$ if $\alpha(i_1) \leq \alpha(i_2)$. We define a linear ordering on $\langle m \rangle$ by $i_1 \preceq i_2$ if

1. $i_1 \in P$ and $i_2 \in Q$,

2. $i_1, i_2 \in P$ and $i_1 \preceq_P i_2$, or

3. $i_1, i_2 \in Q$ and $i_1 \preceq_Q i_2$.

Let σ be permutation associated to the linear ordering $\preceq$ of $\langle m \rangle$, and let σ^{-1} be its inverse. The map $\sigma_*^{-1} : \langle m \rangle_* \to \langle m \rangle_*$ is covered by a tuple morphism

$$g : (s_{\sigma^{-1}(1)}, \ldots, s_{\sigma^{-1}(m)}) \to (s_1, \ldots, s_m),$$

and we define $\mathsf{sort}(f)$ to be the composite

$$\mathsf{sort}(f) = f \circ g.$$

Example 3.1.5.16. The sortings of the morphisms g_1, g_2, and g_3 of Example 3.1.5.13 are shown

below.

g_1 → $\mathsf{sort}(g_1)$

g_2 → $\mathsf{sort}(g_2)$

g_3 → $\mathsf{sort}(g_3)$
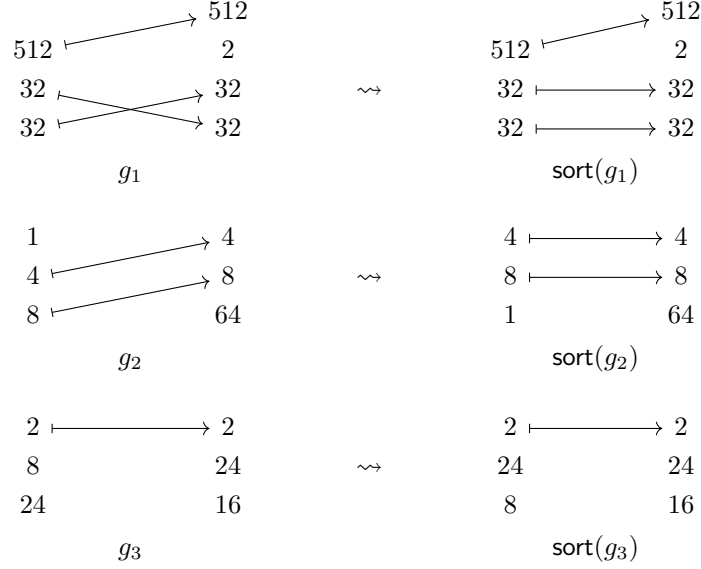
Lemma 3.1.5.17. *Suppose $f : S \to T$ is a tuple morphism. Then f is sorted if and only if $\mathsf{sort}(f) = f$.*

Proof. Our construction of $\mathsf{sort}(-)$ guarantees that $\mathsf{sort}(f)$ is sorted for any tuple morphism f. In particular, if $f = \mathsf{sort}(f)$, then f is sorted. Conversely, if f is sorted, then the permutation $\sigma \in \Sigma_m$ from Construction 3.1.5.15 is the identity permutation, so $g = \mathsf{id}_S$, and so

$$\mathsf{sort}(f) = f \circ \mathsf{id}_S = f.$$

$\square$

Proposition 3.1.5.18. *If f is a tuple morphism, then*

$$L_{\mathsf{sort}(f)} = \mathsf{sort}(L_f).$$

Proof. Borrowing our notation form Construction 3.1.5.15, we have $\mathsf{sort}(f) = f \circ g$ where

$$g : (s_{\sigma^{-1}(1)}, \ldots, s_{\sigma^{-1}(m)}) \to (s_1, \ldots, s_m)$$

lies over $\sigma_*^{-1} : \langle m \rangle_* \to \langle m \rangle_*$. If $L_f = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$, then

$$L_{\mathsf{sort}(f)} = (s_1', \ldots, s_m') : (d_1', \ldots, d_m')$$
$$= (s_{\sigma^{-1}(1)}, \ldots, s_{\sigma^{-1}(m)}) : (d_{\sigma^{-1}(1)}, \ldots, d_{\sigma^{-1}(m)}).$$

Since the modes of $L_{\mathsf{sort}(f)}$ are a permutation of the modes of L_f, it suffices to prove that $L_{\mathsf{sort}(f)}$ is sorted. Suppose $1 \leq i < m$. Suppose first that $\sigma^{-1}(i) \in P$, so that $d_i' = d_{\sigma^{-1}(i)} = 0$. If $\sigma^{-1}(i+1) \in P$, then $d_{i+1}' = d_{\sigma^{-1}(i+1)} = 0$. By construction of σ, we have $s_i' = s_{\sigma^{-1}(i)} \leq s_{\sigma^{-1}(i+1)} = s_{i+1}'$. If instead $\sigma^{-1}(i+1) \in Q$, then $d_{i+1}' = d_{\sigma^{-1}(i+1)}' > 0 = d_i'$. Suppose next that $\sigma^{-1}(i) \in Q$. Then by construction of σ, we have $\sigma^{-1}(i+1) \in Q$ and $\alpha(\sigma^{-1}(i)) < \alpha(\sigma^{-1}(i+1))$, and we have

$$d_i' = d_{\sigma^{-1}(i)} = \prod_{j < \alpha(\sigma^{-1}(i))} t_j$$
$$\leq \prod_{j < \alpha(\sigma^{-1}(i+1))} t_j$$
$$= d_{\sigma^{-1}(i+1)}$$
$$= d_{i+1}',$$

110

where equality holds if and only if $t_{\alpha(\sigma^{-1}(i))} = \cdots = t_{\alpha(\sigma^{-1}(i+1))-1} = 1$. In particular, we have $s_i = s_{\sigma^{-1}(i)} = t_{\alpha(\sigma^{-1}(i))} = 1$, and so $s_i' \leq s_{i+1}'$. We conclude that $L_{\mathsf{sort}(f)}$ is sorted, so $L_{\mathsf{sort}(f)} = \mathsf{sort}(L_f)$. $\square$

Remark 3.1.5.19. The operation $\mathsf{sort}(-)$ is not functorial. For example, consider the tuple morphisms

$$(2,3) \xrightarrow[(2,1)]{f} (3,2) \qquad \text{and} \qquad (10,25) \xrightarrow[(2,1)]{g} (25,10)$$

Then f and g are composable with $g \circ f = \mathsf{id}_{(25,10)}$, but the sorted morphisms

$$(10,25) \xrightarrow[(1,2)]{\mathsf{sort}(f)} (10,25) \qquad \text{and} \qquad (25,10) \xrightarrow[(1,2)]{\mathsf{sort}(g)} (25,10)$$

are not composable.

3.1.5.4 Coalesce

We begin by introducing the notion of a coalesced tuple morphism.

Definition 3.1.5.20. Suppose $f : S \to T$ is a tuple morphism lying over α. We say f is *coalesced* if

1. $S = \mathsf{squeeze}(S)$ and

2. for any $1 \leq i < \mathsf{len}(S)$, exactly one of the following conditions holds:

   (a) $\alpha(i) = * \neq \alpha(i+1)$,
   (b) $\alpha(i) \neq * = \alpha(i+1)$,
   (c) $\alpha(i) > \alpha(i+1)$, or
   (d) $\alpha(i) < \alpha(i+1)$, and there exists $\alpha(i) < j < \alpha(i+1)$ with $t_j > 1$.

Example 3.1.5.21. If there exists some $1 \leq i < \mathsf{len}(S)$ with $\alpha(i+1) = \alpha(i) + 1$, then f is not coalesced.

Remark 3.1.5.22. If $f : S \to T$ is a tuple morphism such that $f = \mathsf{squeeze}(f)$, then f is coalesced if and only if for any $1 \leq i < \mathsf{len}(S)$, one of the following conditions holds:

1. $\alpha(i) = * \neq \alpha(i+1)$,

2. $\alpha(i) \neq * = \alpha(i+1)$,

3. $\alpha(i) > \alpha(i+1)$, or

4. $\alpha(i+1) \neq \alpha(i) + 1$.

Example 3.1.5.23. The morphisms

$$f_1 \qquad\qquad f_2 \qquad\qquad f_3$$

111
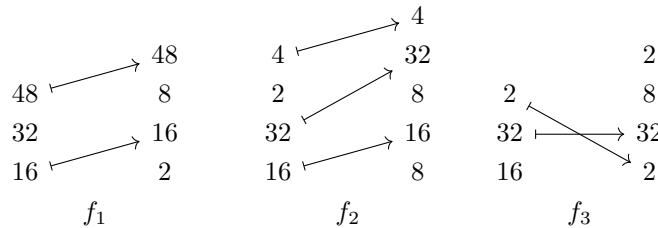
are coalesced, while the morphisms

$$g_1 \qquad\qquad g_2 \qquad\qquad g_3$$
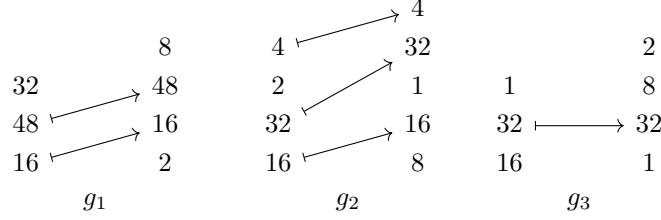
are not coalesced.

Proposition 3.1.5.24. *Suppose f is a tuple morphism. Then f is coalesced if and only if L_f is coalesced.*

Proof. Suppose $f : (s_1, \ldots, s_m) \to (t_1, \ldots, t_n)$ is a tuple morphism, and let

$$L_f = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

be the layout encoded by f.

Suppose first that f is coalesced. Then no entry of $\mathsf{shape}(L_f) = \mathsf{domain}(f)$ is equal to 1. Suppose $1 \leq i < m$. We want to show that $s_i d_i$ is not equal to d_{i+1}. If $d_i = 0$, then $\alpha(i) = *$, and we have

$$
\begin{aligned}
s_i d_i = d_{i+1} \quad &\Leftrightarrow \quad d_{i+1} = 0 \\
&\Leftrightarrow \quad \alpha(i+1) = *
\end{aligned}
$$

but by our assumption that f is coalesced, we have $\alpha(i+1) \neq *$, hence $s_i d_i \neq d_{i+1}$. If $d_i \neq 0$, then $\alpha(i) \neq *$. If $\alpha(i+1) = *$, then $d_{i+1} = 0$, so $s_i d_i \neq d_{i+1}$. If $\alpha(i+1) < \alpha(i)$, then $d_i \geq d_{i+1}$, and since $s_i \neq 1$, we have $s_i d_i > d_{i+1}$. Finally, if $\alpha(i) < \alpha(i+1)$, then

$$
\begin{aligned}
s_i d_i = s_i \cdot \left(\prod_{j < \alpha(i)} t_j \right) = \prod_{j \leq \alpha(i)} t_j \\
< \prod_{j < \alpha(i+1)} t_j \\
= d_{i+1}.
\end{aligned}
$$

We conclude that L_f is coalesced.

Suppose next that the layout L_f is coalesced. Then no entry in $\mathsf{domain}(f) = \mathsf{shape}(L_f)$ is equal to 1. Suppose $1 \leq i < m$. If $\alpha(i) = *$, then $d_i = 0$, and since L_f is coalesced, we must have $d_{i+1} \neq s_i d_i = 0$, hence $\alpha(i+1) \neq *$. Suppose $\alpha(i) \neq *$, and $\alpha(i) < \alpha(i+1)$. Since L_f is coalesced, we have $s_i d_i \neq d_{i+1}$. But if we write

$$s_i d_i = \prod_{j \leq \alpha(i)} t_j,$$

and

$$d_{i+1} = \prod_{j < \alpha(i+1)} t_j,$$

this implies that $\prod_{\alpha(i) < j < \alpha(i+1)} t_j \neq 1$. In particular, there exists some $\alpha(i) < j < \alpha(i+1)$ with $t_j > 1$. We conclude that f is coalesced. $\qquad\square$

Next, we define our $\mathsf{coal}(-)$ operation on tuple morphisms.

Construction 3.1.5.25. Suppose f is a tuple morphism. We define a morphism $\mathsf{coal}(f)$ as follows:

1. First, we set $g = \mathsf{squeeze}(f)$, and we write $\beta : \langle m \rangle_* \to \langle n \rangle_*$ for the map over which g lies.

2. Next, we define an equivalence relation $\sim$ on $\langle m \rangle$ where $i \sim i'$ if either

   (a) $\beta(i'') = *$ for $i \le i'' \le i'$, or

   (b) $\beta(i'') = \beta(i) + (i'' - i)$ for $i \le i'' \le i'$.

   The quotient $\langle m \rangle / \sim$ is ordered by $[i_1] \le [i_2]$ if $i_1 \le i_2$, so we can identify this quotient with $\langle \bar{m} \rangle$ where $\bar{m}$ is the size of $\langle m \rangle / \sim$.

3. Next, define an equivalence relation $\sim$ on $\langle n \rangle$ where $j \sim j'$ if there exists $i \in \langle m \rangle$ such that

$$\beta(i + (j'' - j)) = \beta(i) + (j'' - j)$$

   for all $j \le j'' \le j'$. The quotient $\langle n \rangle / \sim$ is ordered by $[j_1] \le [j_2]$ if $j_1 \le j_2$, so we can identify this quotient with $\langle \bar{n} \rangle$ where $\bar{n}$ is the size of $\langle n \rangle / \sim$.

4. Next, we observe that the map $\beta : \langle m \rangle_* \to \langle n \rangle_*$ descends to a map

$$\bar{\beta} : \langle \bar{m} \rangle_* \to \langle \bar{n} \rangle_*$$

   given by $\bar{\beta}([i]) = [\beta(i)]$.

5. The domain $\bar{S} = (\bar{s}_1, \ldots, \bar{s}_{\bar{m}})$ of $\mathsf{coal}(f)$ is defined by setting

$$\bar{s}_i = \prod_{i' \in I} s_{i'}$$

   if $i \in \langle \bar{m} \rangle$ corresponds to the equivalence class $I \in \langle m \rangle / \sim$. The codomain $\bar{T} = (\bar{t}_1, \ldots, \bar{t}_{\bar{n}})$ of $\mathsf{coal}(f)$ is defined by setting

$$\bar{t}_j = \prod_{j' \in J} t_{j'}$$

   if $j \in \langle \bar{n} \rangle$ corresponds to the equivalence class $J \in \langle n \rangle / \sim$. We then define

$$\mathsf{coal}(f) : \bar{S} \to \bar{T}$$

   to be the tuple morphism lying over $\bar{\beta}$.

Example 3.1.5.26. Here is an example of a tuple morphism f and the coalesced morphism $\mathsf{coal}(f)$.

113
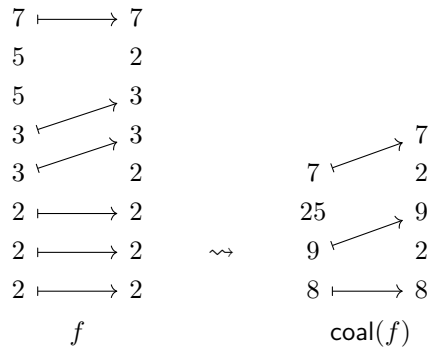
Example 3.1.5.27. We can coalesce the morphism f of Example 3.1.5.8 as follows
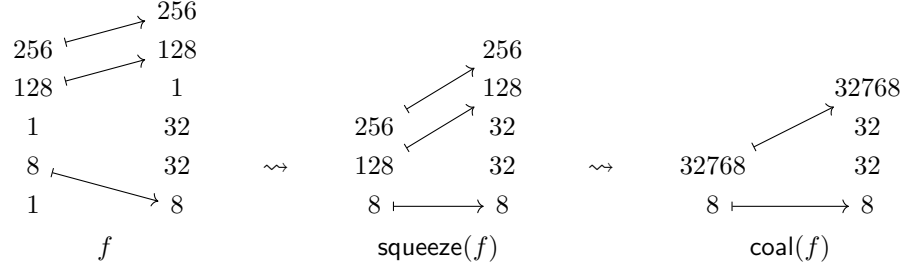
$$f \qquad\qquad \mathsf{squeeze}(f) \qquad\qquad \mathsf{coal}(f)$$

Proposition 3.1.5.28. *If f is a tuple morphism, then*

1. $\mathsf{coal}(f)$ *is coalesced, and*

2. $L_{\mathsf{coal}(f)} = \mathsf{coal}(L_f)$.

Proof. First, we will argue that $\mathsf{coal}(f)$ is coalesced. This is immediate from our construction, since applying $\mathsf{squeeze}$ eliminates all modes equal to 1, and passing to the quotient in our construction consolidated all adjacent modes with $\alpha(i + 1) = \alpha(i) + 1$.

Next, we will prove that $L_{\mathsf{coal}(f)} = \mathsf{coal}(L_f)$. In light of Proposition 2.1.4.18 and Proposition 3.1.5.24, it suffices to prove that $\Phi_{\mathsf{coal}(f)} = \Phi_f$. Certainly applying $\mathsf{squeeze}(-)$ to f has no impact on the associated layout function, so we need to argue that passing to the quotient in our construction does not change the layout function of the associated layout. This follows from the fact that forming our quotient can be formed in steps, where in each step we combine adjacent modes with either $\alpha(i) = * = \alpha(i + 1)$, or $\alpha(i + 1) = \alpha(i) + 1$. These correspond to replacing adjacent modes of the form $s_i, s_{i+1} : 0, 0$ with $s_i s_{i+1} : 0$, and $s_i, s_{i+1} : d_i, s_i d_i$ with $s_i s_{i+1} : d_i$, respectively. Neither such operation changes the layout function of a layout, and so we conclude that $\Phi_{L_{\mathsf{coal}(f)}} = \Phi_{\mathsf{coal}(L_f)}$, as desired. $\qquad\square$

3.1.5.5   Concatenate

Next, we will define a concatenation operation on tuple morphisms. This operation may be performed on tuple morphisms satisfying a "disjointness" condition, which we specify below.

Definition 3.1.5.29. Suppose $\alpha : \langle m \rangle_* \to \langle n \rangle_*$ and $\beta : \langle p \rangle_* \to \langle n \rangle_*$ are morphisms in $\mathbf{Fin}_*$ with the same codomain. We say α and β have *disjoint images* if

$$\mathsf{Image}(\alpha) \cap \mathsf{Image}(\beta) = \{*\}.$$

Construction 3.1.5.30. If $\alpha : \langle m \rangle_* \to \langle n \rangle_*$ and $\beta : \langle p \rangle_* \to \langle n \rangle_*$ have disjoint images, then we have a well-defined morphism

$$\alpha \star \beta : \langle m + p \rangle_* \to \langle n \rangle_*$$

given by

$$(\alpha \star \beta)(i) = \begin{cases} * & i = * \\ \alpha(i) & 1 \leq i \leq m \\ \beta(i - m) & m + 1 \leq i \leq m + p. \end{cases}$$

This operation is associative, so we can consider $\alpha_1 \star \cdots \star \alpha_k$ for any collection of morphisms $\alpha_1, \ldots, \alpha_k$ in $\mathbf{Fin}_*$ with pairwise disjoint images.

Remark 3.1.5.31. If α and β are tractable pointed maps and α and β have disjoint images, then $\alpha \star \beta$ is tractable.
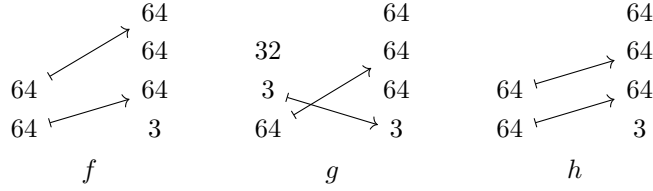
Definition 3.1.5.32. Suppose

$$f : S \to T$$

and

$$g : U \to T$$

are tuple morphisms lying over α and β, respectively. We say f and g have *disjoint images* if the morphisms α and β have disjoint images.

Example 3.1.5.33. Consider the tuple morphisms f, g, and h shown below.

Then f and g have disjoint images, while h and g do not have disjoint images.

Construction 3.1.5.34. Suppose

$$f : S \to T, \text{ and } g : U \to T$$

are tuple morphisms lying over α and β, respectively, and that f and g have disjoint images. We define the *concatenation* of f and g to be the morphism

$$f \star g : S \star U \to T$$

lying over $\alpha \star \beta$. This operation is associative, so we can consider $f_1 \cdots f_k$ for any finite collection of morphisms f_i with pairwise disjoint images.

Example 3.1.5.35. If f and g are the morphisms in **Tuple** from Example 3.1.5.33, then the concatenation of f and g is the morphism shown below.

$$f \star g$$
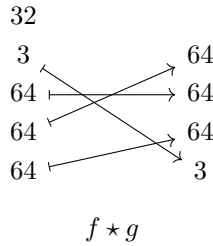
Example 3.1.5.36. Suppose $f : (s_1, \ldots, s_m) \to (t_1, \ldots, t_n)$ is a tuple morphism, and for any $1 \leq i \leq m$, let

$$f_i : (s_i) \to (t_1, \ldots, t_n)$$

denote the ith entry of f, as in Example 3.1.3.8. Then we can write

$$f = f_1 \star \cdots \star f_m$$

as the concatenation of its entries.

Lemma 3.1.5.37. *Suppose $f_1 : S_1 \to T$ and $f_2 : S_2 \to T$ are tuple morphisms with disjoint images. If $g : T \to U$ is any tuple morphism, then*

$$g \circ (f_1 \star f_2) = (g \circ f_1) \star (g \circ f_2).$$

Proof. Suppose f_1, f_2, and g lie over $\alpha_1 : \langle m_1 \rangle_* \to \langle n \rangle$, $\alpha_2 : \langle m_2 \rangle_* \to \langle n \rangle$, and $\beta : \langle n \rangle \to \langle p \rangle$, respectively. The two maps in question have the same domains and the same codomains, so it suffices to prove that

$$\beta \circ (\alpha_1 \star \alpha_2) = (\beta \circ \alpha_1) \star (\beta \circ \alpha_2).$$

We compute

$$
\begin{aligned}
(\beta \circ (\alpha_1 \star \alpha_2))(i) &= \beta((\alpha_1 \star \alpha_2)(i)) \\
&= \begin{cases} \beta(*) & i = * \\ \beta(\alpha_1(i)) & 1 \leq i \leq m_1 \\ \beta(\alpha_2(i - m_1)) & m_1 + 1 \leq i \leq m_1 + m_2 \end{cases} \\
&= \begin{cases} * & i = * \\ (\beta \circ \alpha_1)(i) & 1 \leq i \leq m_1 \\ (\beta \circ \alpha_2)(i - m_1) & m_1 + 1 \leq i \leq m_1 + m_2 \end{cases} \\
&= ((\beta \circ \alpha_1) \star (\beta \circ \alpha_2))(i).
\end{aligned}
$$

$\square$

Proposition 3.1.5.38. *Suppose $f_1, \ldots, f_k$ are morphisms in **Tuple** with the same codomain and with pairwise disjoint images. Then the layouts $L_{f_1}, \ldots, L_{f_k}$ satisfy*

$$L_{f_1 \star \cdots \star f_k} = L_{f_1} \star \cdots \star L_{f_k}.$$

Proof. First, we prove the result for $k = 2$. Suppose

$$f = (s_1, \ldots, s_m) \to (t_1, \ldots, t_n), \text{ and } g : (u_1, \ldots, u_p) \to (t_1, \ldots, t_n)$$

have disjoint images, and write

$$L_f = (s_1, \ldots, s_m) : (d_1, \ldots, d_m), \text{ and } L_g = (u_1, \ldots, u_p) : (d'_1, \ldots, d'_p).$$

Then the layout $L_{f \star g}$ is given by

$$L_{f \star g} = (s_1, \ldots, s_m, u_1, \ldots, u_p) : (e_1, \ldots, e_{m+m'})$$

where

$$
\begin{aligned}
e_i &= \prod_{j < (\alpha \star \beta)(i)} t_j \\
&= \begin{cases} \displaystyle\prod_{j < \alpha(i)} t_j & 1 \leq i \leq m \\ \displaystyle\prod_{j < \beta(i-m)} t_j & m + 1 \leq i \leq m + m'. \end{cases} \\
&= \begin{cases} d_i & 1 \leq i \leq m \\ d'_{i-m} & m + 1 \leq i \leq m + m'. \end{cases}
\end{aligned}
$$

This concludes the proof of the result when $k = 2$. The general case follows from the associativity of concatenation of tuple morphisms, and the associativity of concatenation of flat layouts. $\square$

3.1.5.6 Complement

We begin by defining the notion of complementary tuple morphisms.

Definition 3.1.5.39. Suppose $f : S \to T$ and $g : U \to T$ are tuple morphisms. We say g is a *complement* of f if

1. f and g have disjoint images, and

2. the concatenation

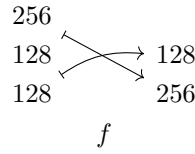$$f \star g : S \star U \xrightarrow{\cong} T$$

is an isomorphism.

Example 3.1.5.40. If f and g are the morphisms shown below
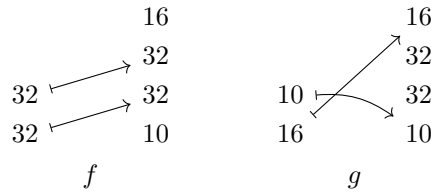
then g is a complement of f.

Example 3.1.5.41. If f is the morphism shown below

then f does not admit a complement.

Next, we prove that complementary tuple morphisms give rise to complementary flat layouts.

Proposition 3.1.5.42. *If $f : S \to T$ is a tuple morphism and g is a complement of f, then L_g is a* $\mathsf{size}(T)$*-complement of L_f.*

Proof. Write $S = \mathsf{domain}(f)$, $U = \mathsf{domain}(g)$, and $T = \mathsf{codomain}(f) = \mathsf{codomain}(g)$. First, we note that

$$
\begin{aligned}
\mathsf{size}(L_f) \cdot \mathsf{size}(L_g) &= \mathsf{size}(L_f \star L_g) \\
&= \mathsf{size}(L_{f \star g}) \\
&= \mathsf{size}(S \star U) \\
&= \mathsf{size}(T).
\end{aligned}
$$

Next, we note that $f \star g$ is an isomorphism, hence so is

$$|f \star g| = \Phi_{L_{f \star g}}^{\mathsf{size}(T)}$$

where we have used the identification of $\Phi_{L_{f \star g}}^{\mathsf{size}(T)}$ of Lemma 3.1.4.5. $\square$

117

Proposition 3.1.5.43. *If f is an injective tuple morphism, then*

$$\mathsf{coal}^\flat(L_{f^c}) = \mathsf{comp}^\flat(L_f, \mathsf{size}(T)).$$

Proof. By Proposition 3.1.5.42, we know that L_{f^c} is a $\mathsf{size}(T)$-complement of L_f. Since f^c is sorted, so is L_{f^c} and it follows from Proposition 2.1.6.33, it follows that

$$\mathsf{coal}^\flat(L_{f^c}) = \mathsf{comp}^\flat(L_f, \mathsf{size}(T)),$$

since both of these layouts are flat, sorted, coalesced complements of L_f of the same size. □

Proposition 3.1.5.44. *If $f : (s_1, \ldots, s_m) \to (t_1, \ldots, t_n)$ is an injective tuple morphism of standard form, then*

$$L_{f^c} = \mathsf{comp}^\flat(L_f).$$

Proof. Write

$$L_f = (s_1, \ldots, s_m) : (d_1, \ldots, d_m)$$

for the layout encoded by f. By Proposition 3.1.5.42, we know that L_{f^c} is a $\mathsf{size}(T)$-complement of L_f. Where

$$\mathsf{size}(T) = t_1 \cdots t_n = (t_1 \cdots t_{n-1})t_n$$
$$= d_m s_m.$$

By construction, f^c is sorted, hence so is L_{f^c}. Moreover, since f has standard form, it follows that f^c is coalesced. By Proposition 2.1.6.23, we deduce that

$$L_{f^c} = \mathsf{comp}^\flat(L_f).$$

□

Definition 3.1.5.45. Suppose f is a tuple morphism lying over $\alpha : \langle m \rangle_* \to \langle n \rangle_*$. We say f is *complementable* if α is injective.

Construction 3.1.5.46. Suppose $f : (s_1, \ldots, s_m) \to (t_1, \ldots, t_n)$ is a complementable tuple morphism. Let $j_1 < \cdots < j_{n-m}$ denote the collection of indices in $\langle n \rangle$ which are not in the image of α. We define the *complement of f* to be the tuple morphism

$$f^c : (t_{j_1}, \ldots, t_{j_k}) \to (t_1, \ldots, t_n)$$

lying over the map $\mathsf{complement}(\alpha) : \langle n - m \rangle_* \to \langle n \rangle_*$ given by $k \mapsto j_k$. By construction, we may observe that f^c is a complement of f, in the sense of Definition 3.1.5.39

Example 3.1.5.47. Below is an example of a morphism f and its complement f^c.

Proposition 3.1.5.48. *If f is a tuple morphism and g is a complement of f, then*

$$\mathsf{sort}(g) = f^c.$$

Proof. Suppose f lies over $\alpha : \langle m \rangle_* \to \langle n \rangle_*$, $\mathsf{sort}(g)$ lies over $\beta : \langle n - m \rangle_* \to \langle n \rangle_*$ and f^c lies over $\alpha^c : \langle n - m \rangle_* \to \langle n \rangle_*$. Then β and α^c are increasing maps with the same image, namely

$$\mathsf{Image}(\beta) = \langle n \rangle \setminus \mathsf{Image}(\alpha) = \mathsf{Image}(\alpha^c),$$

hence $\beta = \alpha^c$, and hence $\mathsf{sort}(g) = f^c$. $\square$

Proposition 3.1.5.49. *Suppose f is a tuple morphism. Then f admits a complement if and only if f is complementable, in the sense of Definition 3.1.5.45.*

Proof. If f lies over a map α which is not injective, then for any morphism f^* such that f and f^* have disjoint images, the morphism $f \star f^*$ lies over a map which is not injective, hence $f \star f^*$ is not an isomorphism. Conversely, if f lies over an injective map, then the morphism f^c of Construction 3.1.5.46 is a complement of f. $\square$

Proposition 3.1.5.50. *If f is a complementable tuple morphism, then*

$$\mathsf{sort}(f) = (f^c)^c.$$

Proof. Both maps are increasing, injective, and have the same image, so they are equal. $\square$

3.1.5.7 Flat division

In this section, we define a division operation on tuple morphisms.

Definition 3.1.5.51. If f and g are tuple morphisms, we say g *divides* f if g and f are composable. In other words,

$$\mathsf{codomain}(g) = \mathsf{domain}(f).$$

Definition 3.1.5.52. Suppose $g : S \to T$ and $f : T \to U$ are tuple morphisms. The *flat division* of f by g is the tuple morphism

$$f \oslash^\flat g = f \circ (g \star g^c).$$

Example 3.1.5.53. Here is an example of tuple morphisms f and g together with their flat quotient $f \oslash^\flat g$.
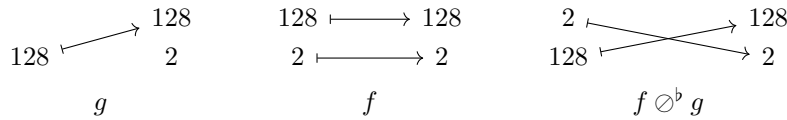
Example 3.1.5.54. Here is an example of tuple morphisms f and g together with their flat quotient $f \oslash^\flat g$.
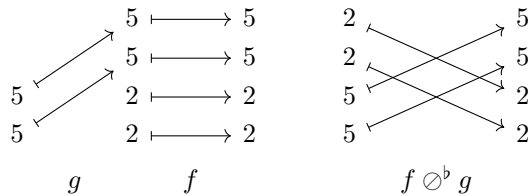
Example 3.1.5.55. Here is an example of tuple morphisms f and g together with their flat quotient $f \oslash^\flat g$.
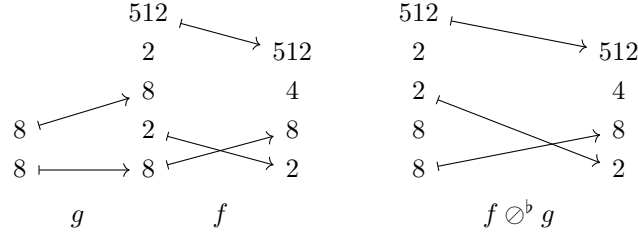
Proposition 3.1.5.56. *If f and g are non-degenerate composable tuple morphisms, then*

$$\mathsf{coal}^\flat(L_{f \oslash^\flat g}) = \mathsf{coal}^\flat(L_f \oslash^\flat L_g)$$

Proof. By Proposition 3.2.6.20, we have

$$\mathsf{coal}^\flat(L_{g^c}) = \mathsf{comp}^\flat(L_g, \mathsf{size}(L_f)),$$

and we compute

$$
\begin{aligned}
\mathsf{coal}^\flat(L_f \oslash^\flat L_g) &= \mathsf{coal}^\flat(L_f \circ (L_g \star \mathsf{comp}(L_g, \mathsf{size}(L_f)))) \\
&= \mathsf{coal}(L_f \circ (L_g \star L_{g^c})) \\
&= \mathsf{coal}(L_f \circ L_{g \star g^c}) \\
&= \mathsf{coal}(L_{f \circ (g \star g^c)}) \\
&= \mathsf{coal}(L_{f \oslash^\flat g}).
\end{aligned}
$$

$\square$

3.1.5.8 Flat products

In this section we define a product operation on tuple morphisms.

Definition 3.1.5.57. Suppose f and g are tuple morphisms. We say f and g are *product admissible* if $\mathsf{codomain}(g) = \mathsf{domain}(f^c)$. If f and g are product admissible, then we define *flat product* of f and g to be

$$f \otimes^\flat g = f \star (f^c \circ g).$$

Example 3.1.5.58. If f and g are the tuple morphisms shown below
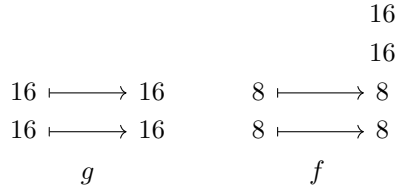
then f and g are product-admissible, and $f \otimes^\flat g$ is the tuple morphism shown below.
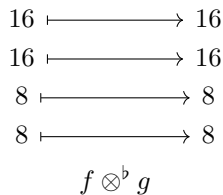
Example 3.1.5.59. If f and g are the tuple morphisms shown below
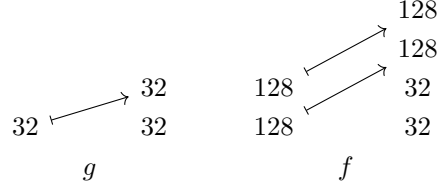
then f and g are product-admissible, and $f \otimes^\flat g$ is the tuple morphism shown below.

$$f \otimes^\flat g$$
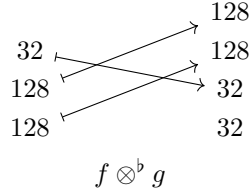
Lemma 3.1.5.60. *If f and g are product admissible and g is injective, then $f \otimes^\flat g$ is injective and*

$$(f \otimes^\flat g)^c = f^c \circ g^c.$$

Proof. The tuple morphisms $(f \otimes^\flat g)^c$ and $f^c \circ g^c$ are injective, increasing, and have the same codomain, so it suffices to show that they have the same image. The image of $(f \otimes^\flat g)^c = (f \star (f^c \circ g))^c$ consists of those entries which are not in the image of f, and not in the image of $f^c \circ g$. The image of f^c consists of those entries which are not in the image of f, and so the image of the composition $f^c \circ g^c$ consists of those entries which are not in the image of f, and not in the image of $f^c \circ g$. $\square$

Proposition 3.1.5.61. *Suppose f and g are product admissible, and g and h are product admissible. Then*

1. *$f \otimes^\flat g$ and h are product admissible,*

2. *f and $g \otimes^\flat h$ are product admissible, and*

3. *$(f \otimes^\flat g) \otimes^\flat h = f \otimes^\flat (g \otimes^\flat h)$.*

Proof. Using Lemma 3.1.5.37 and Lemma 3.1.5.60, we compute

$$
\begin{aligned}
f \otimes^\flat (g \otimes^\flat h) &= f \star (f^c \circ (g \otimes^\flat h)) \\
&= f \star (f^c \circ (g \star (g^c \circ h))) \\
&= f \star ((f^c \circ g) \star (f^c \circ (g^c \circ h))) \\
&= f \star ((f^c \circ g) \star ((f^c \circ g^c) \circ h)) \\
&= f \star (f^c \circ g) \star ((f \otimes^\flat g)^c \circ h) \\
&= (f \otimes^\flat g) \star ((f \otimes^\flat g)^c \circ h) \\
&= (f \otimes^\flat g) \otimes^\flat h.
\end{aligned}
$$

$\square$

Proposition 3.1.5.62. *Suppose f and g are non-degenerate tuple morphisms and that f and g are product admissible. Then*

$$L_{f \otimes^\flat g} = L_f \otimes^\flat L_g.$$

Proof. Suppose $f : S \to T$ and $g : U \to V$ are product admissible, and set

$$L_f^* = \mathsf{comp}^\flat(L_f, \mathsf{size}(L_f) \cdot \mathsf{cosize}(L_g)).$$

Since f is injective and the codomain of g is the domain of f^c, it follows that

$$\mathsf{size}(L_f) \cdot \mathsf{cosize}(L_g) \le \mathsf{size}(S) \cdot \mathsf{size}(V) = \mathsf{size}(T).$$

Using this fact, and the fact that

$$\Phi_{\mathsf{comp}(L_f, \mathsf{size}(T))} = \Phi_{L_{f^c}},$$

we have

$$\begin{aligned}
L_f^* \circ L_g &= \mathsf{comp}(L_f, \mathsf{size}(T)) \circ L_g \\
&= L_{f^c} \circ L_g.
\end{aligned}$$

Using this fact, we compute

$$\begin{aligned}
L_f \otimes^\flat L_g &= L_f \star (L_f^* \circ L_g) \\
&= L_f \star (L_{f^c} \circ L_g) \\
&= L_f \star L_{f^c \circ g} \\
&= L_{f \star (f^c \circ g)} \\
&= L_{f \otimes^\flat g}
\end{aligned}$$

$\square$

3.2 The category **Nest**

In the previous section, we introduced a category **Tuple**, whose morphisms encode flat tractable layouts. In this section, we introduce a category **Nest**, whose morphisms encode tractable layouts with arbitrary nesting.

3.2.1 Basic definitions

Recall that for a nested tuple S, we write $S^\flat$ for the flattening of S. For example, if $S = (64, (8, 8))$, then $S^\flat = (64, 8, 8)$.

Definition 3.2.1.1. Let **Nest** denote the category whose objects are nested tuples of positive integers, and in which a morphism

$$f : S \to T$$

in **Nest** is specified by a tuple morphism

$$f^\flat : S^\flat \to T^\flat.$$

In other words,

$$\mathsf{Hom}_{\mathbf{Nest}}(S, T) = \mathsf{Hom}_{\mathbf{Tuple}}(S^\flat, T^\flat).$$

Explicitly, a morphism $f : S \to T$ in **Nest** is specified by a tractable pointed map $\alpha : \langle \mathsf{len}(S) \rangle_* \to \langle \mathsf{len}(T) \rangle_*$ satisfying the following property:

- If $1 \le i \le \mathsf{len}(S)$ and $\alpha(i) \neq *$, then $\mathsf{entry}_i(S) = \mathsf{entry}_{\alpha(i)}(T)$.

We say such a morphism f *lies over* α, and refer to f as a *nested tuple morphism*.

Notation 3.2.1.2. If $f : S \to T$ is a nested tuple morphism which lies over α, we depict f as

$$S \xrightarrow[\alpha]{f} T$$

Example 3.2.1.3. Here are some examples of nested tuple morphisms.

$$(64, (8, 8)) \xrightarrow[(1,2,3)]{f} (64, 8, 8)$$

$$((2, 2), 2) \xrightarrow[(*,5,2)]{g} (10, 2, 2, (3, 2, 3))$$

$$64 \xrightarrow[(2)]{h} ((64, 64), 512).$$

Observation 3.2.1.4. If X is a set, lets write X^{ind} for the indiscrete category on X. This is the category whose objects are the elements of X, and in which there is a unique (iso)morphism between any two objects. Then by definition of **Nest**, we have a pullback square

$$
\begin{array}{ccc}
\mathbf{Nest} & \xrightarrow{\mathsf{prof}(-)} & \mathsf{Profile}^{\mathsf{ind}} \\
{\scriptstyle (-)^\flat} \downarrow & \lrcorner & \downarrow {\scriptstyle \mathsf{len}(-)} \\
\mathbf{Tuple} & \xrightarrow[\mathsf{len}(-)]{} & \mathbb{N}^{\mathsf{ind}}
\end{array}
$$

We may view this as a categorification of the pullback square 2.2.2.4.

Example 3.2.1.5. Suppose S is a nested tuple of length m. If $1 \le i \le m$ then there is a nested tuple morphism

$$\mathsf{entry}_i(S) \to S$$

lying over the map $\langle 1 \rangle_* \to \langle m \rangle_*$ given by $1 \mapsto i$. For instance, if $S = (64, (8, 8))$ and $i = 1$, then we have a nested tuple morphism

$$64 \xrightarrow{\quad (1) \quad} (64, (8, 8)).$$

Example 3.2.1.6. Suppose S is a nested tuple of rank r. If $1 \le i \le r$, then there is a canonical nested tuple morphism

$$\mathsf{mode}_i(S) \to S$$

lying over the map $\langle \mathsf{len}_i(S) \rangle_* \to \langle \mathsf{len}(S) \rangle_*$ given by $j \mapsto j + \mathsf{len}_{<i}(S)$. For instance, if $S = (64, (8, 8))$, then we have a nested tuple morphism

$$(8, 8) \xrightarrow{\quad (2,3) \quad} (64, (8, 8)).$$

Observation 3.2.1.7. There are functors relating the categories **Nest** and **Tuple**. First, there is an inclusion functor

$$\textbf{Tuple} \xrightarrow{\ \subset\ } \textbf{Nest}$$

which considers a tuple morphism $f : S \to T$ as a nested tuple morphism. Next, there is a *flattening* functor

$$\textbf{Nest} \xrightarrow{\ (-)^\flat\ } \textbf{Tuple}$$

which sends a nested tuple morphism $f : S \to T$ to the underlying tuple morphism $f^\flat : S^\flat \to T^\flat$. The composite

$$\textbf{Tuple} \xrightarrow{\ \subset\ } \textbf{Nest} \xrightarrow{\ (-)^\flat\ } \textbf{Tuple}$$

is the identity functor on **Tuple**, so **Tuple** is a retractive subcategory of **Nest**. Moreover, these functors form an adjoint equivalence of categories.

Remark 3.2.1.8. One might wish to consider some category **C** whose morphisms encode tractable layouts, but which is *not* equivalent to **Tuple**. The authors have considered several such examples, but leave their investigation to future work.

3.2.2   From nested tuple morphisms to layouts

The key feature of the category **Nest** is that if $f : S \to T$ is a nested tuple morphism, then f encodes a layout L_f. This layout is obtained by equipping the flat layout $L_{f^\flat}$ with the nesting profile of S. More precisely, we have the following construction.

Construction 3.2.2.1. Suppose

$$f : S \to T$$

is a nested tuple morphism, and suppose $P = \mathsf{prof}(S)$. We define L_f to be the layout

$$L_f = (L_{f^\flat})_P$$

where $(-)_P$ is the P-substitution operation of Definition 2.3.1.19. We refer to L_f as the *layout encoded by f*.

Construction 3.2.2.2. Suppose

$$(s_1, \ldots, s_m)_P \xrightarrow[\alpha]{f} (t_1, \ldots, t_n)_Q$$

is a nested tuple morphism. We define L_f to be the layout whose shape

$$\mathsf{shape}(L_f) = (s_1, \ldots, s_m)_P$$

is the domain of f, and whose stride

$$\mathsf{stride}(L_f) = (d_1, \ldots, d_m)_P$$

has entries defined by the formula

$$d_i = \begin{cases} 0 & \alpha(i) = * \\ \prod_{j < \alpha(i)} t_j & \alpha(i) \neq *. \end{cases}$$

We refer to L_f as the *layout encoded by f*.

Example 3.2.2.3. The layout encoded by

$$((8, 8), (4, 4)) \xrightarrow[(1,4,3,2)]{f} (8, 4, 4, 8)$$

is

$$L_f = ((8, 8), (4, 4)) : ((1, 128), (32, 8)).$$

Example 3.2.2.4. The layout encoded by

$$(128, (4, 4, 2)) \xrightarrow[(3,1,2,*)]{g} ((4, 4), 128)$$

is

$$L_g = (128, (4, 4, 2)) : (16, (1, 4, 0)).$$

Observation 3.2.2.5. The flattening functor

$$\mathsf{Nest} \xrightarrow{(-)^\flat} \mathsf{Tuple}$$

is compatible with flattening of layouts, in that if f is a nested tuple morphism, then

$$(L_f)^\flat = L_{f^\flat}.$$

If L is a tractable layout, then we can construct a nested tuple morphism which encodes L as follows.

Construction 3.2.2.6. Suppose L is a tractable layout. We define the *standard representation* of L to be the nested tuple morphism

$$f_L : S \to T$$

where $(f_L)^\flat = f_{L^\flat}$ is the standard representation of $L^\flat$, $S = \mathsf{shape}(L)$ is the shape of L, and T is the codomain of $f_{L^\flat}$.

Example 3.2.2.7. If
$$L = (32, (2, 2)) : (192, (24, 3))$$
then the standard representation of L is
$$(32, (2, 2)) \xrightarrow[(6,4,2)]{f_L} (3, 2, 4, 2, 4, 32).$$

Lemma 3.2.2.8. *If L is a tractable layout, and $f = f_L$ is the standard representation of L, then*
$$L_f = L.$$

Proof. We have
$$(L_f)^\flat = L_{f^\flat} = L^\flat$$
and
$$\mathsf{shape}(L_f) = \mathsf{shape}(L).$$

$\square$

Proposition 3.2.2.9. *Suppose L is a layout. Then there exists a nested tuple morphism f encoding L if and only if L is tractable.*

Proof. Suppose first that $L = L_f$ for some nested tuple morphism f. Then $(L_f)^\flat = L_{f^\flat}$, and by Proposition 3.1.2.10, we know that $L^\flat$ is tractable, hence so is L. Conversely, if L is tractable, then we can take $f = f_L$ to be the standard representation of L, and by Lemma 3.2.2.8, we have $L_f = L$. $\square$

In order to establish a one-to-one correspondence between tractable layouts and certain nested tuple morphisms, we introduce the notion of *standard form* for nested tuple morphisms.

Definition 3.2.2.10. Suppose $f : S \to T$ is a nested tuple morphism. We say f has *standard form* if

1. $f^\flat$ has standard form, as in Definition 3.1.2.12, and

2. T is flat.

Example 3.2.2.11. The nested tuple morphism
$$((2, 2), (3, 3)) \xrightarrow[(4,6,2,3)]{f} (10, 3, 3, 2, 10, 2)$$
has standard form.

Example 3.2.2.12. The nested tuple morphism
$$((2, 2), (3, 3)) \xrightarrow[(4,6,2,3)]{f} ((10, 3, 3), (2, 10, 2))$$
does not have standard form since the codomain of g is not flat.

Just as in the flat case, we need to exclude non-degenerate nested tuple morphisms and non-degenerate layouts in order to obtain a one-to-one correspondence between nested tuple morphisms of standard form and tractable layouts. To this end, we make the following definition.

126

Definition 3.2.2.13. Suppose

$$S \xrightarrow[\alpha]{f} T$$

is a nested tuple morphism, and suppose

$$L = S : D$$

is a layout.

1. We say f is *non-degenerate* if

$$\mathsf{entry}_i(S) = 1 \quad \Rightarrow \quad \alpha(i) = *.$$

2. We say L is *non-degenerate* if

$$\mathsf{entry}_i(S) = 1 \quad \Rightarrow \quad \mathsf{entry}_i(D) = 0.$$

Remark 3.2.2.14. If f is a nested tuple morphism, then f is non-degenerate if and only if $f^\flat$ is non-degenerate. If L is a layout, then L is non-degenerate if and only if $L^\flat$ is non-degenerate.

Proposition 3.2.2.15. *The maps*

$$f \longmapsto L_f$$

$$\left\{ \begin{array}{c} \textit{Non-degenerate} \\ \textit{nested tuple morphisms} \\ \textit{of standard form} \end{array} \right\} \longleftrightarrow \left\{ \begin{array}{c} \textit{Non-degenerate} \\ \textit{tractable layouts} \end{array} \right\}$$

$$f_L \longleftarrow\!\mid L$$

of Constructions 3.2.2.2 and 3.2.2.6 determine a one-to-one correspondence between nested tuple morphisms of standard form, and tractable layouts.

Proof. We have already shown in Proposition 3.2.2.9 that if L is a tractable layout and $f = f_L$ is the standard form of L, then $L_f = L$. Suppose next that f has standard form, and let $L = L_f$ be the layout encoded by f. We want to show that f is equal to the standard representation f_L of L. By Proposition 3.1.2.21, we know that $f^\flat$ is equal to the standard representation $f_{L^\flat}$ of $L^\flat$, and since

$$\mathsf{domain}(f) = \mathsf{shape}(L) = \mathsf{domain}(f_L),$$

and

$$\mathsf{codomain}(f) = \mathsf{codomain}(f^\flat) = \mathsf{codomain}(f_{L^\flat}) = \mathsf{codomain}(f_L),$$

we deduce that $f = f_L$. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

3.2.3 Examples

In this section, we list some important families of nested tuple morphisms.

Example 3.2.3.1 (Reparenthesizations)**.** Suppose S_1 and S_2 are nested tuples with the same flattening

$$S_1^\flat = S_2^\flat.$$

Then there is a *reparenthesization isomorphism*

$$\mathsf{id}_{S_1}^{S_2} : S_1 \xrightarrow{\cong} S_2$$

lying over the identity. These morphisms are transitive, in that

$$\mathsf{id}_{S_2}^{S_3} \circ \mathsf{id}_{S_1}^{S_2} = \mathsf{id}_{S_1}^{S_3},$$

and compatible with identities, in that

$$\mathsf{id}_S^S = \mathsf{id}_S.$$

If $f = \mathsf{id}_{S_1}^{S_2}$ is a reparenthesization isomorphism, then L_f is the column major layout with shape S_1.

Example 3.2.3.2 (Flattenings)**.** As a special case of the previous example, if S is any nested tuple, then we have a flattening isomorphism

$$\mathsf{id}_S^{S^\flat} : S \xrightarrow{\cong} S^\flat$$

and an unflattening isomorphism

$$\mathsf{id}_{S^\flat}^S : S^\flat \xrightarrow{\cong} S$$

Observation 3.2.3.3. If $f : S \to T$ is a nested tuple, then f is equal to the composite

$$S \xrightarrow{\mathsf{id}_S^{S^\flat}} S^\flat \xrightarrow{f^\flat} T^\flat \xrightarrow{\mathsf{id}_{T^\flat}^T} T.$$

$$\underbrace{\hspace{4cm}}_{f}$$

In other words, we have a canonical factorization

$$f = \mathsf{id}_{T^\flat}^T \circ f^\flat \circ \mathsf{id}_S^{S^\flat}.$$

Example 3.2.3.4 (Entries)**.** Suppose

$$S \xrightarrow[\alpha]{f} T$$

is a nested tuple morphism. Suppose $1 \leq i \leq \mathsf{len}(S)$, and write $j = \alpha(i)$. Then we refer to the nested tuple morphism

$$\mathsf{entry}_i(S) \xrightarrow[(j)]{\mathsf{entry}_i(f)} T$$

as the ith entry of f. The layout encoded by $\mathsf{entry}_i(f)$ is

$$L_{\mathsf{entry}_i(f)} = \mathsf{entry}_i(L_f).$$

Example 3.2.3.5 (Entry inclusions)**.** As a special case of the previous example, if S is a nested tuple and $1 \leq i \leq \mathsf{len}(S)$, we can take $f = \mathsf{id}_S$, in which case

$$\mathsf{entry}_i(\mathsf{id}_S) : \mathsf{entry}_i(S) \longrightarrow S$$

is the inclusion of the ith entry of S.

128

Example 3.2.3.6 (Modes)**.** Suppose

$$S \xrightarrow[\alpha]{f} T$$

is a nested tuple morphism. Suppose $1 \leq i \leq \mathsf{rank}(S)$ and, write

$$N = \mathsf{len}_{<i}(S)$$
$$\ell = \mathsf{len}_i(S).$$

Then we refer to the nested tuple morphism

$$\mathsf{mode}_i(S) \xrightarrow[(N+1,\ldots,N+\ell)]{\mathsf{mode}_i(f)} T$$

as the ith mode of S. The layout encoded by $\mathsf{mode}_i(L_f)$ is

$$L_{\mathsf{mode}_i(f)} = \mathsf{mode}_i(L_f).$$

Example 3.2.3.7 (Mode inclusions)**.** As a special case of the previous example we may take $f = \mathsf{id}_S$, in which case

$$\mathsf{mode}_i(\mathsf{id}_S) : \mathsf{mode}_i(S) \to S$$

is the inclusion of the ith mode of S. We sometime denote this map by

$$\mathsf{incl}_i(S) = \mathsf{mode}_i(\mathsf{id}_S).$$

3.2.4 Realization of nested tuple morphisms

In the flat case, we constructed a realization functor

$$\mathsf{Tuple} \xrightarrow{|\cdot|} \mathsf{FinSet}$$

which sends a tuple morphism f to the layout function of L_f. We can extend this to a realization functor

$$\mathsf{Nest} \xrightarrow{|\cdot|} \mathsf{FinSet}$$

by precomposing with the flattening functor $\mathsf{Nest} \to \mathsf{Tuple}$.

Definition 3.2.4.1. We define the *realization functor*

$$\mathsf{Nest} \xrightarrow{|\cdot|} \mathsf{FinSet}$$

to be the composite

$$\mathsf{Nest} \xrightarrow{(-)^\flat} \mathsf{Tuple} \xrightarrow{|\cdot|} \mathsf{FinSet}$$

Lemma 3.2.4.2. *If $f : S \to T$ is a nested tuple morphism, then the realization $|f|$ of f is the layout function of L_f:*

$$|f| = \Phi_{L_f}^{\mathsf{size}(T)}.$$

Proof. This follows immediately from 3.1.4.5, since

$$|f| = |f^\flat| = \Phi_{L_{f^\flat}}^{\mathsf{size}(T)} = \Phi_{L_f}^{\mathsf{size}(T)}$$

$$\square$$

129

3.2.5 Refinements

In this section, we revisit the refinement of nested tuples from a categorical perspective. Recall from section 2.2.4 that a nested tuple S' *refines* S, denoted

$$S' \longrightarrow\!\!\!\!\!\rightarrow S$$

if S' may be obtained from S by replacing each entry of S with some nested tuple of the same size. For example,

$$(2, (2,2)) \twoheadrightarrow 8,$$

and

$$((2,2),(3,3),(5,5)) \twoheadrightarrow (4,9,25).$$

If $\mathsf{len}(S) = m$ and $\mathsf{prof}(S) = P$, then we can write

$$S' = (S'_1, \ldots, S'_m)_P$$

as the P-substitution of the relative modes

$$S'_i = \mathsf{mode}_i(S', S).$$

We refer to the ordinary concatenation

$$(S'_1, \ldots, S'_m) = \mathsf{flat}(S', S)$$

as the flattening of S' relative to S.

Let **Ref** denote the poset category of nested tuples of positive integers under refinement, so that a morphism in **Ref** is a refinement $S' \twoheadrightarrow S$. If S is a nested tuple, let

$$\mathbf{Ref}(S) = \{S' \mid S' \text{ refines } S\}$$

denote the poset of nested tuples refining S. Equivalently, $\mathbf{Ref}(S)$ is the slice category $\mathbf{Ref}_{/S}$.

Construction 3.2.5.1. [Relative mode inclusions] Suppose $S' \twoheadrightarrow S$ is a refinement, and write

$$S'_i = \mathsf{mode}_i(S', S)$$

for the modes of S' relative to S. Then S' and $(S'_1, \ldots, S'_m)$ have the same flattening, so we have a reparenthesization isomorphism

$$\mathsf{id}^{S'}_{(S'_1, \ldots, S'_m)} : (S'_1, \ldots, S'_m) \xrightarrow{\ \cong\ } S'$$

and we define

$$\mathsf{incl}_i(S', S) : S'_i \to S'$$

to be the composite

$$S'_i \xrightarrow{\ \mathsf{incl}_i((S'_1, \ldots, S'_m))\ } (S'_1, \ldots, S'_m) \xrightarrow{\ \mathsf{id}^{S'}_{(S'_1, \ldots, S'_m)}\ } S'$$

of the ith mode inclusion of $(S'_1, \ldots, S'_m)$ with the reparenthesization isomorphism $(S'_1, \ldots, S'_m) \cong S'$.

Example 3.2.5.2. If $S = (4, (9, 25))$ and $S' = ((2,2), ((3,3), 25))$, then S' refines S, and $\mathsf{incl}_2(S', S)$ is the nested tuple morphism

$$(3,3) \xrightarrow[\substack{(3,4)}]{\ \mathsf{incl}_2(S', S)\ } ((2,2), ((3,3), 25)).$$

Construction 3.2.5.3. [Relative modes] Suppose $f' : S' \to T'$ is a nested tuple morphism, and suppose S' refines S. We define the *ith mode of f' relative to S*, denoted

$$\mathsf{mode}_i(f', S) = f' \circ \mathsf{incl}_i(S', S) : S'_i \to T'$$

to be the composite

$$S'_i \xrightarrow{\mathsf{incl}_i(S',S)} S' \xrightarrow{f'} T'$$

In particular, we have

$$\mathsf{mode}_i(\mathsf{id}_{S'}, S) = \mathsf{incl}_i(S', S).$$

Example 3.2.5.4. Suppose $S = (4, (9, 25))$ and $S' = ((2, 2), ((3, 3), 25))$, so that S' refines S. If f' is the nested tuple morphism

$$((2, 2), ((3, 3), 25)) \xrightarrow[(1,3,2,*,4)]{f'} (2, 3, 2, 25).$$

then $\mathsf{mode}_2(f', S)$ is the nested tuple morphism

$$(3, 3) \xrightarrow[(2,*)]{\mathsf{mode}_2(f',S)} (2, 3, 2, 25).$$

Construction 3.2.5.5 (Pullbacks). Suppose $f : S \to T$ is a nested tuple morphism lying over α, and suppose $T' \twoheadrightarrow T$ is a refinement. Let

$$T'_j = \mathsf{mode}_j(T', T)$$

denote the jth mode of T' relative to T, and for any $1 \leq i \leq \mathsf{len}(S)$, set

$$S'_i = \begin{cases} \mathsf{entry}_i(S) & \alpha(i) = * \\ T'_j & \alpha(i) = j. \end{cases}$$

We define the *pullback of T' along f* to be the nested tuple

$$S' = f^* T' = \mathsf{sub}(S, (S'_1, \ldots, S'_m)).$$

For any $1 \leq i \leq m$, let

$$f'_i : S'_i \to T'$$

be the trivial map if $\alpha(i) = *$, and the inclusion

$$\mathsf{incl}_j(T', T) : S'_i = T'_j \to T'$$

if $\alpha(i) = j$. The maps $f'_1, \ldots, f'_m$ have disjoint images, so we form the concatenation

$$(f'_1, \ldots, f'_m) : (S'_1, \ldots, S'_m) \to T'.$$

We define $f' = T'^* f$ to be the composite

$$S' \xrightarrow{\mathsf{id}_{S'}^{(S'_1, \ldots, S'_m)}} (S'_1, \ldots, S'_m) \xrightarrow{(f'_1, \ldots, f'_m)} T'.$$

We refer to f' as the pullback of f along T, and depict such a pullback as a square

$$\begin{array}{ccc} S' & \xrightarrow{f'} & T' \\ \downarrow & \lrcorner & \downarrow \\ S & \xrightarrow{f} & T. \end{array}$$

Example 3.2.5.6. Suppose $f : (64, 32) \to (4, 64, 4, 32)$ lies over $\alpha = (2, 4)$. Then we have a pullback square

$$
\begin{array}{ccc}
((16, 4), (16, 2)) & \xrightarrow{\ f'\ } & ((2, 2), (16, 4), (2, 2), (16, 2)) \\
\downarrow & \lrcorner & \downarrow \\
(64, 32) & \xrightarrow[\ f\ ]{} & (4, 64, 4, 32)
\end{array}
$$

where f' lies over $\alpha' = (3, 4, 7, 8)$.

Example 3.2.5.7. Suppose S is a nested tuple with flattening

$$S^\flat = (s_1, \ldots, s_m),$$

and suppose $S' \twoheadrightarrow S$ is a refinement with relative flattening

$$(S'_1, \ldots, S'_m).$$

Then the pullback of $S' \twoheadrightarrow S$ along the unflattening isomorphism

$$\mathsf{id}^S_{(s_1, \ldots, s_m)} : (s_1, \ldots, s_m) \to S$$

is the reparenthesization isomorphism

$$
\begin{array}{ccc}
(S'_1, \ldots, S'_m) & \xrightarrow{\ \mathsf{id}^{S'}_{(S'_1, \ldots, S'_m)}\ } & S' \\
\downarrow & \lrcorner & \downarrow \\
(s_1, \ldots, s_m) & \xrightarrow[\ \mathsf{id}^S_{(s_1, \ldots, s_m)}\ ]{} & S.
\end{array}
$$

Example 3.2.5.8. Suppose $S' \twoheadrightarrow S$ is a refinement, and consider the ith entry inclusion

$$s_i \to S.$$

Then the pullback of $S' \twoheadrightarrow S$ along $s_i \to S$ is the ith relative mode inclusion

$$
\begin{array}{ccc}
S'_i & \xrightarrow{\ \mathsf{incl}_i(S', S)\ } & S' \\
\downarrow & \lrcorner & \downarrow \\
s_i & \xrightarrow{\quad\quad} & S.
\end{array}
$$

Observation 3.2.5.9. The pullback construction above specifies a contravariant functor

$$\mathbf{Nest}^{\mathrm{op}} \xrightarrow{\quad\quad} \mathbf{Cat}$$

$$
\begin{array}{ccccc}
S & \longmapsto & \mathbf{Ref}(S) & \quad & f^*T' \leftarrow f^*T'' \\
f\downarrow & & \uparrow f^* & & \uparrow \\
T & \longmapsto & \mathbf{Ref}(T) & & T' \leftarrow T''
\end{array}
$$

The key property of pullbacks is that the layout function of f' is equal to that of f.

132

Lemma 3.2.5.10. *Suppose*

$$S' \xrightarrow{f'} T'$$
$$\downarrow \quad \lrcorner \quad \downarrow$$
$$S \xrightarrow{f} T$$

is a pullback square, where f lies over α. Let

$$f'_i : S'_i \to T$$

denote the ith mode of f' relative to S, and let

$$(L_f)^\flat = (s_1, \ldots, s_m) : (d_1, \ldots, d_m).$$

Then for any $1 \le i \le m$, we have

$$\mathsf{coal}(L_{f'_i}) = s_i : d_i.$$

Proof. Suppose $1 \le i \le m$. If $\alpha(i) = *$, then f'_i is the trivial map, so

$$L_{f'_i} = s_i : 0 = s_i : d_i.$$

In particular, $\mathsf{coal}(L_{f'_i}) = s_i : 0 = s_i : d_i$. Suppose next that $\alpha(i) = j \ne *$. By construction of f', we have that

$$f'_i = \mathsf{incl}_j(T', T) : T'_j \to T'.$$

which lies over the map α'_i given by $t \mapsto \mathsf{len}_{<j}(T', T) + t$. For each $1 \le t < \mathsf{len}(T'_j)$, we have $\alpha'_i(t) = \alpha'_i(t+1)$, so $L_{f'_i}$ is a column major layout with size $\mathsf{size}(T'_j) = t_j = s_i$. This implies that $\mathsf{coal}(L_{f'_i})$ is a depth 0 layout of the form

$$\mathsf{coal}(L_{f'_i}) = s_i : e$$

for some integer $e \ge 0$. We claim that $e = d_i$. If we write $t'_{j'} = \mathsf{entry}_{j'}(T')$, then we have

$$\begin{aligned}
e = \mathsf{entry}_1(\mathsf{stride}(L_{f'_i})) &= \prod_{j' < \alpha'_i(1)} t'_{j'} \\
&= \prod_{j' \le \mathsf{len}_{<j}(T', T)} t'_{j'} \\
&= \prod_{j' < j} \mathsf{size}(T'_{j'}) \\
&= \prod_{j' < j} t_{j'} \\
&= d_i.
\end{aligned}$$

$\square$

Proposition 3.2.5.11. *If*

$$S' \xrightarrow{f'} T'$$
$$\downarrow \quad \lrcorner \quad \downarrow$$
$$S \xrightarrow{f} T$$

is a pullback square, then $\Phi_{L_f} = \Phi_{L_{f'}}$.

Proof. We begin by fixing notation. Let $m = \mathsf{len}(S)$, and let

$$S^\flat = (s_1, \ldots, s_m),$$
$$S'_i = \mathsf{mode}_i(S', S),$$
$$T'_j = \mathsf{mode}_j(T', T),$$
$$(L_f)^\flat = (s_1, \ldots, s_m) : (d_1, \ldots, d_m).$$

Consider the reparenthesization isomorphism

$$\mathsf{id}^{S'}_{(S'_1, \ldots, S'_m)} : (S'_1, \ldots, S'_m) \to S'$$

The composite of this map with f' is the concatenation $(f'_1, \ldots, f'_m)$ where f'_i is the trivial map if $\alpha(i) = *$, and the relative mode inclusion

$$\mathsf{incl}_i(T', T) : S'_i = T'_j \to T'$$

otherwise. Using Lemma 3.2.5.10, and the fact that $L_{f'} = L_{(f'_1, \ldots, f'_m)}$, we compute

$$\begin{aligned}
\mathsf{coal}(L_{f'}) &= \mathsf{coal}(L_{(f'_1, \ldots, f'_m)}) \\
&= \mathsf{coal}((L_{f'_1}, \ldots, L_{f'_m})) \\
&= \mathsf{coal}((\mathsf{coal}(L_{f'_1}), \ldots, \mathsf{coal}(L_{f'_m}))) \\
&= \mathsf{coal}((s_1, \ldots, s_m) : (d_1, \ldots, d_m)) \\
&= \mathsf{coal}(L_f).
\end{aligned}$$

By Proposition 2.3.3.14, we deduce that $\Phi_{L_{f'}} = \Phi_{L_f}$. $\qquad\square$

Construction 3.2.5.12 (Pushforwards)**.** Suppose $f : S \to T$ is a nested tuple morphism lying over α, and suppose $S' \twoheadrightarrow S$ is a refinement. Let

$$S'_i = \mathsf{mode}_i(S', S)$$

denote the ith mode of S' relative to S, and for any $1 \le j \le \mathsf{len}(T)$, set

$$T'_j = \begin{cases} \mathsf{entry}_j(T) & j \notin \mathsf{Image}(\alpha) \\ S'_i & \alpha(i) = j. \end{cases}$$

We define the *pushforward of S' along f* to be the nested tuple

$$T' = f_* S' = \mathsf{sub}(T, (T'_1, \ldots, T'_n)).$$

For any $1 \le i \le m$, let

$$f'_i : S'_i \to T'$$

be the trivial map if $\alpha(i) = *$, and the relative mode inclusion

$$\mathsf{incl}_j(T', T) : S'_i = T'_j \to T'$$

if $\alpha(i) = j$. The morphisms $f'_1, \ldots, f'_m$ have disjoint images, so we can form the concatenation

$$(f'_1, \ldots, f'_m) : (S'_1, \ldots, S'_m) \to T'.$$

We define $f' = S'_* f$ to be the composite

$$S' \xrightarrow{\ \mathsf{id}_{S'}^{(S'_1,\dots,S'_m)}\ } (S'_1,\dots,S'_m) \xrightarrow{\ (f'_1,\dots,f'_m)\ } T'.$$

We refer to f' as the pushforward of f along T. We depict such a pushforward as

$$
\begin{array}{ccc}
S' & \xrightarrow{\ f'\ } & T' \\
\downarrow & \lrcorner & \downarrow \\
S & \xrightarrow{\ f\ } & T
\end{array}
$$

Example 3.2.5.13. If $f : (64, 32) \to (4, 64, 4, 32)$ lies over $\alpha = (2, 4)$, then we have a pushforward square

$$
\begin{array}{ccc}
((16,4),(16,2)) & \xrightarrow{\ f'\ } & (4,(16,4),4,(16,2)) \\
\downarrow & \lrcorner & \downarrow \\
(64,32) & \xrightarrow{\ f\ } & (4,64,4,32)
\end{array}
$$

The key property of pullbacks is that the layout function of f' is equal to that of f.

Lemma 3.2.5.14. *Suppose*

$$
\begin{array}{ccc}
S' & \xrightarrow{\ f'\ } & T' \\
\downarrow & \lrcorner & \downarrow \\
S & \xrightarrow{\ f\ } & T
\end{array}
$$

is a pushforward square, where f lies over α. Let

$$f'_i : S'_i \to T$$

denote the ith mode of f' relative to S, and let

$$(L_f)^\flat = (s_1,\dots,s_m) : (d_1,\dots,d_m).$$

Then for any $1 \le i \le m$, we have

$$\mathsf{coal}(L_{f'_i}) = s_i : d_i.$$

Proof. The proof is identical to that of Lemma 3.2.5.10 □

Proposition 3.2.5.15. *If*

$$
\begin{array}{ccc}
S' & \xrightarrow{\ f'\ } & T' \\
\downarrow & \lrcorner & \downarrow \\
S & \xrightarrow{\ f\ } & T
\end{array}
$$

is a pushforward square, then $\Phi_{L_f} = \Phi_{L_{f'}}$.

Proof. The proof is identical to that of Proposition 3.2.5.11. □

Observation 3.2.5.16. The pushforward construction defined above specifies a covariant functor

$$\mathbf{Nest} \longrightarrow \mathbf{Cat}$$

$$
\begin{array}{ccc}
S \longmapsto \mathbf{Ref}(S) & \qquad S'' \to S' \\
f\downarrow \qquad \qquad \downarrow f_* & \qquad \qquad \downarrow \\
T \longmapsto \mathbf{Ref}(T) & \qquad f_*S'' \to f_*S'
\end{array}
$$

Observation 3.2.5.17. If $f : S \to T$ is an isomorphism of nested tuples, then

$$\mathbf{Ref}(T) \xrightarrow{\ f^*\ } \mathbf{Ref}(S)$$

and

$$\mathbf{Ref}(S) \xrightarrow{\ f_*\ } \mathbf{Ref}(T)$$

are inverse isomorphisms of categories. Specifically,

$$(f^{-1})^* = f_* \quad \text{and} \quad (f^{-1})_* = f^*.$$

Observation 3.2.5.18. If S_1 and S_2 are nested tuples with $\mathsf{flat}(S_1) = \mathsf{flat}(S_2)$, then there is a canonical nested tuple isomorphism $S_1 \cong S_2$, and hence, a canonical isomorphism of categories

$$\mathbf{Ref}(S_1) \cong \mathbf{Ref}(S_2).$$

There is one more concept we need to specify, called *mutual refinements*. The importance of this concept will be come clear in Chapter 4, when we use this concept in our layout composition algorithm.

Definition 3.2.5.19. Suppose T and U are nested tuples. A *mutual refinement* of (T, U) is a diagram of the form

$$
\begin{array}{ccc}
T' & \rightarrowtail & U' \\
\downarrow & & \downarrow \\
T & & U
\end{array}
$$

Explicitly, this is a pair of nested tuples (T', U') such that

1. T' refines T,

2. U' refines U, and

3. T' divides U'.

In addition to the definition of mutual refinements, we need the following fact.

Lemma 3.2.5.20. *Suppose T and U are nested tuples. Then there is a one-to-one correspondence between mutual refinements of (T, U), and mutual refinements of $(T^\flat, U^\flat)$.*

Proof. If (T', U') is a mutual refinement of (T, U), then pulling back along the unflattening isomorphisms $\mathsf{id}_{T^\flat}^T$ and $\mathsf{id}_{U^\flat}^U$ yields a mutual refinement

$$
\begin{array}{ccc}
(\mathsf{id}_{T^\flat}^T)^*T' & \rightarrowtail & (\mathsf{id}_{U^\flat}^U)^*U' \\
\downarrow & & \downarrow \\
T^\flat & & U^\flat
\end{array}
$$

of $(T^\flat, U^\flat)$. Conversely, if $((T^\flat)', (U^\flat)')$ is a mutual refinement of $T^\flat, U^\flat$, then pulling back along the flattening isomorphisms $\mathsf{id}_T^{T^\flat}$ and $\mathsf{id}_U^{U^\flat}$ yields a mutual refinement

$$(\mathsf{id}_T^{T^\flat})^*(T^\flat)' \rightarrowtail (\mathsf{id}_U^{U^\flat})^*(U^\flat)'$$
$$\downarrow \qquad\qquad \downarrow$$
$$T \qquad\qquad U$$

of $(T^\flat, U^\flat)$. $\qquad\qquad\qquad\square$

3.2.6 Operations on nested tuple morphisms

Our next task is to develop an "algebra of nested tuple morphisms". Since we have already developed such an "algebra" for tuple morphisms, we can extend to the nested case by equipping the outputs of our various operations with an appropriate profile.

3.2.6.1 Concatenate

Next, we define a concatenation operation on nested tuple morphisms, which is compatible with concatenation of layouts, in that
$$L_{(f,g)} = (L_f, L_g).$$
We concatenate nested tuple morphisms f and g by concatenating the domains of f and g. In order for this to be well-defined, we need f and g to satisfy a disjointness condition, which we specify below.

Definition 3.2.6.1. Suppose f and g are nested tuple morphisms with the same codomain. We say f and g have *disjoint images* if $f^\flat$ and $g^\flat$ have disjoint images, as in Definition 3.1.5.32.

Example 3.2.6.2. If
$$f : (3, (512, 512)) \to (2, 512, 2, 512)$$
lies over $(*, 2, 4)$ and
$$g : (2, 2) \to (2, 512, 2, 512)$$
lies over $(1, 3)$, then f and g have disjoint images.

Example 3.2.6.3. If
$$f : (2, (32, 64)) \to (32, (2, 2, 2), 64)$$
lies over $\alpha = (3, 1, 5)$ and
$$g : ((2, 2)) \to (32, (2, 2, 2), 64)$$
lies over $\beta = (2, 4)$, then f and g have disjoint images.

Construction 3.2.6.4. Suppose $f : S \to T$ and $g : U \to T$ are nested tuple morphisms lying over α and β, respectively, and that f and g have disjoint images. We define the *concatenation* of f and g to be the nested tuple morphism
$$(f, g) : (S, U) \to T$$
with

$$\mathsf{flat}((f, g)) = f^\flat \star g^\flat.$$

More generally, if $f_i : S_i \to T$ are nested tuple morphisms for $1 \le i \le k$, and $f_1, \ldots, f_k$ have pairwise disjoint images, then we define the *concatenation*

$$(f_1, \ldots, f_k) : (S_1, \ldots, S_k) \to T.$$

to be the nested tuple morphism with

$$(f_1, \ldots, f_k)^\flat = f_1^\flat \star \cdots \star f_k^\flat.$$

Example 3.2.6.5. The concatenation of the morphisms f and g of Example 3.2.6.2 is the nested tuple morphism
$$(f, g) : ((3, (512, 512)), (2, 2)) \to (2, 512, 2, 512)$$
lying over $\alpha \star \beta = (*, 2, 4, 1, 3)$.

Example 3.2.6.6. The concatenation of the morphisms f and g of Example 3.2.6.3 is the nested tuple morphism

$$(f, g) : ((2, (32, 64)), ((2, 2))) \to (32, (2, 2, 2), 64)$$

lying over $\alpha \star \beta = (3, 1, 5, 2, 4)$.

Example 3.2.6.7. If

$$f : (2, 2) \to (2, 3, 5, 2, 3, 5)$$

lies over $\alpha = (1, 4)$

$$g : (3, 3) \to (2, 3, 5, 2, 3, 5)$$

lies over $\beta = (2, 5)$, and

$$h : (5, 5) \to (2, 3, 5, 2, 3, 5)$$

lies over $\gamma = (3, 6)$, then f, g and h have pairwise disjoint images, and the concatenation

$$(f, g, h) : ((2, 2), (3, 3), (5, 5)) \to (2, 3, 5, 2, 3, 5)$$

lies over $\alpha \star \beta \star \gamma = (1, 4, 2, 5, 3, 6)$.

Example 3.2.6.8. Suppose $f : S \to T$ is a nested tuple morphism, and suppose

$$S^\flat = (s_1, \dots, s_m).$$

Recall from example 3.2.3.4 that for any $1 \le i \le m$, there is a nested tuple morphism

$$f_i : s_i \to T.$$

called the ith entry of f. These morphisms have pairwise disjoint images, and the concatenation

$$(f_1, \dots, f_m) : S^\flat \to T$$

is the composite

$$(f_1, \dots, f_m) = f \circ \mathsf{id}_{S^\flat}^S$$

of Example 3.2.3.2

Example 3.2.6.9. Suppose $f : S \to T$ is a nested tuple morphism, and suppose

$$S = (S_1, \dots, S_r).$$

Recall from example 3.2.3.6 that for any $1 \le i \le r$, there is a nested tuple morphism

$$f_i : S_i \to T.$$

called the ith mode of f. These morphisms have pairwise disjoint images, and the concatenation

$$(f_1, \dots, f_r) : S \to T$$

is equal to f. In other words, every nested tuple morphism f may be written as the concatenation of its modes:

$$f = (f_1, \dots, f_r).$$

Proposition 3.2.6.10. *If $f_1, \dots, f_k$ are nested tuple morphisms with the same codomain and with pairwise disjoint images, then*

$$L_{(f_1, \dots, f_k)} = (L_{f_1}, \dots, L_{f_k}).$$

139

Proof. By construction, we have

$$\mathsf{shape}((L_{f_1}, \ldots, L_{f_k})) = (\mathsf{shape}(L_{f_1}), \ldots, \mathsf{shape}(L_{f_k}))$$
$$= \mathsf{shape}(L_{(f_1, \ldots, f_k)}).$$

and using Proposition 3.1.5.38, we have

$$(L_{f_1}, \ldots, L_{f_k})^\flat = L_{f_1}^\flat \star \cdots \star L_{f_k}^\flat$$
$$= L_{f_1^\flat} \star \cdots \star L_{f_k^\flat}$$
$$= L_{f_1^\flat \star \cdots \star f_k^\flat}$$
$$= L_{(f_1, \ldots, f_k)^\flat}$$
$$= (L_{(f_1, \ldots, f_k)})^\flat.$$

$\square$

3.2.6.2 Coalesce

If f is a nested tuple morphism, then we might define $\mathsf{coal}(f)$ to be $\mathsf{coal}^\flat(f^\flat)$. Theoretically, this is a sound definition. However, in order to make our definitions compatible with the `cute` implementation, we make a small modification to our definition of $\mathsf{coal}(f)$.

Definition 3.2.6.11. Suppose $f : S \to T$ is a nested tuple morphism, and write

$$\mathsf{coal}^\flat(f^\flat) : (s_1, \ldots, s_m) \to (t_1, \ldots, t_n).$$

- (Case 1): If $m > 1$, we define
$$\mathsf{coal}(f) = \mathsf{coal}^\flat(f^\flat).$$

- (Case 2): If $m = 1$, we define $\mathsf{coal}(f)$ to be the composite

$$s_1 \xrightarrow[(1)]{} (s_1) \xrightarrow{\mathsf{coal}^\flat(f^\flat)} (t_1, \ldots, t_n).$$

- (Case 3): If $m = 0$, we define $\mathsf{coal}(f)$ to be the composite

$$1 \xrightarrow[(*)]{} () \xrightarrow{\mathsf{coal}^\flat(f^\flat)} (t_1, \ldots, t_n).$$

Example 3.2.6.12. If
$$f : ((2,2), (3,3), (5,5)) \to (5,5,3,3,2,2)$$

lies over $\alpha = (5, 6, 3, 4, 1, 2)$, then

$$\mathsf{coal}(f) : (4, 9, 25) \to (25, 9, 4)$$

lies over $\alpha' = (3, 2, 1)$.

Proposition 3.2.6.13. *If $f : S \to T$ is a nested tuple morphism, then*

$$\mathsf{coal}(L_f) = L_{\mathsf{coal}(f)}.$$

Proof. Let's again write

$$(s_1, \ldots, s_m) \xrightarrow[\alpha]{\mathsf{coal}^\flat(f^\flat)} (t_1, \ldots, t_n).$$

There are three cases to consider.

- (Case 1): Suppose $m > 1$. Then

$$\begin{aligned}
L_{\mathsf{coal}(f)} &= L_{\mathsf{coal}^\flat(f^\flat)} \\
&= \mathsf{coal}^\flat(L_{f^\flat}) \\
&= \mathsf{coal}((L_f)^\flat) \\
&= \mathsf{coal}(L_f).
\end{aligned}$$

- (Case 2): Suppose $m = 1$. Then

$$\begin{aligned}
L_{\mathsf{coal}(f)} &= s_1 : t_1 \ldots, t_{\alpha(1)-1} \\
&= \mathsf{coal}((s_1) : (t_1 \cdots t_{\alpha(1)-1})) \\
&= \mathsf{coal}(L_{\mathsf{coal}^\flat(f^\flat)}) \\
&= \mathsf{coal}(\mathsf{coal}^\flat(L_{f^\flat})) \\
&= \mathsf{coal}((L_f)^\flat) \\
&= \mathsf{coal}(L_f).
\end{aligned}$$

- (Case 3): Suppose $m = 0$. Then

$$\begin{aligned}
L_{\mathsf{coal}(f)} &= 1 : 0 \\
&= \mathsf{coal}(() : ()) \\
&= \mathsf{coal}(L_{\mathsf{coal}^\flat(f^\flat)}) \\
&= \mathsf{coal}(\mathsf{coal}^\flat(L_{f^\flat})) \\
&= \mathsf{coal}((L_f)^\flat) \\
&= \mathsf{coal}(L_f).
\end{aligned}$$

$\square$

3.2.6.3 Complement

In this section, we define the notion of complementary nested tuple morphisms.

Definition 3.2.6.14. Suppose $f : S \to T$ and $g : U \to T$ are nested tuple morphisms with disjoint images. We say g is a *complement* of f if

$$(f, g) : (S, U) \to T$$

is an isomorphism.

Remark 3.2.6.15. If $f : S \to T$ and $g : U \to T$ are nested tuple morphisms, then g is a complement of f if and only if $g^\flat$ is a complement of $f^\flat$, since $(f, g)^\flat = f^\flat \star g^\flat$.

Proposition 3.2.6.16. *If $f : S \to T$ is a nested tuple morphism and $g : U \to T$ is a complement of f, then L_g is a $\mathsf{size}(T)$-complement of L_f.*

Proof. Observation 3.2.2.5 implies that

$$(L_f)^\flat = L_{f^\flat}, \text{ and}$$
$$(L_g)^\flat = L_{g^\flat}$$

and Lemma 2.3.6.2 allows us to reduce to the flat case (Proposition 3.1.5.42). □

Construction 3.2.6.17. Suppose $f : S \to T$ is a nested nested tuple morphism. We define the *complement of f* to be the composite

where $(f^\flat)^c$, is as defined in Construction 3.1.5.46, and $\mathsf{id}_{T^\flat}^T : T^\flat \cong T$ is the unflattening isomorphism.

Example 3.2.6.18. The complement of the nested tuple morphism

$$((2,2),(5,5)) \xrightarrow[\;(1,4,2,5)\;]{f} ((2,5,7),(2,5,7))$$

is

$$(7,7) \xrightarrow[\;(3,6)\;]{f^c} ((2,5,7),(2,5,7)).$$

Proposition 3.2.6.19. *Suppose $f : S \to T$ and $g : U \to T$ are nested tuple morphisms. If f is injective and g is a complement of f, then L_g is a $\mathsf{size}(T)$-complement of L_f.*

Proof. This follows from Proposition 3.1.5.42 and Lemma 2.3.6.2 since

$$(L_f)^\flat = L_{f^\flat}$$
$$(L_g)^\flat = L_{g^\flat}.$$

□

Proposition 3.2.6.20. *If $f : S \to T$ is an injective nested tuple morphism, then*

$$\mathsf{coal}(L_{f^c}) = \mathsf{comp}(L_f, \mathsf{size}(T)).$$

Proof. Since f^c is obtained from $(f^\flat)^c$ by post-composing with a reparenthesization isomorphism, it follows that

$$L_{f^c} = L_{(f^\flat)^c}$$

so by Proposition 3.2.6.20, it follows that

$$\mathsf{coal}^\flat(L_{f^c}) = \mathsf{comp}^\flat(L_f, \mathsf{size}(T)).$$

Applying $\mathsf{coal}(-)$ to both sides yields the result. □

3.2.6.4 Composition

We can use the realization functor of Section 3.2.4 to prove that composition of nested tuple morphisms is compatible with composition of the associated layouts.

Theorem 3.2.6.21. *If f and g are non-degenerate composable nested tuple morphisms, then*

$$L_{g \circ f} = L_g \circ L_f.$$

Proof. Suppose $f : S \to T$ and $g : T \to U$ are non-degenerate nested tuple morphisms. We need to check that

1. $\mathsf{shape}(L_{g \circ f})$ *refines* $\mathsf{shape}(L_f)$: This holds since

$$\mathsf{shape}(L_f) = S = \mathsf{shape}(L_{g \circ f}).$$

2. $L_{g \circ f}$ *is coalesced over* $\mathsf{shape}(L_f)$: This holds since the nested tuple morphism $g \circ f$ is non-degenerate, hence so is the layout $L_{g \circ f}$.

3. $\Phi_{L_{g \circ f}} = \Phi_{L_g} \circ \Phi_{L_f}^{\mathsf{size}(L_g)}$: Using Lemma 3.2.4.2, we have

$$\begin{aligned}
\Phi_{L_{g \circ f}}^{\mathsf{size}(U)} &= |g \circ f| \\
&= |g| \circ |f| \\
&= \Phi_{L_g}^{\mathsf{size}(U)} \circ \Phi_{L_f}^{\mathsf{size}(T)}
\end{aligned}$$

and by postcomposing with the inclusion $[0, \mathsf{size}(U)) \subset \mathbb{Z}$, and observing that $\mathsf{size}(T) = \mathsf{size}(L_g)$, the result follows.

$\square$

3.2.6.5 Logical division

Next, we introduce logical division of nested tuple morphisms. This construction is obtained from flat division by introducing nesting profiles, with no compatibility constraints.

Definition 3.2.6.22. Suppose f and g are nested tuple morphisms. We say g *divides* f if g and f are composable. In other words,

$$\mathsf{codomain}(g) = \mathsf{domain}(f).$$

Definition 3.2.6.23. Suppose $g : S \to T$ and $f : T \to U$ are nested tuple morphisms. We define the logical division of f by g to be the nested tuple morphism

$$f \oslash g = f \circ (g, g^c).$$

Example 3.2.6.24. The logical division of

$$((2,2),2) \xrightarrow[\;(2,4,*)\;]{f} ((4,2),(4,2))$$

by

$$(2,2) \xrightarrow[\;(1,3)\;]{g} ((2,2),2)$$

is

$$((2,2),2) \xrightarrow[\;(2,*,4)\;]{f \oslash g} ((4,2),(4,2)).$$

Example 3.2.6.25. The logical division of

$$(8, 8, 512, 512, 512) \xrightarrow[(*,*,1,2,3)]{f} (512, 512, 512)$$

by

$$(8, 512) \xrightarrow[(1,5)]{g} (8, 8, 512, 512, 512)$$

is

$$((8, 512), (8, 512, 512)) \xrightarrow[(*,1,*,2,3)]{f \oslash g} ((4, 2), (4, 2)).$$

Proposition 3.2.6.26. *If $g : S \to T$ and $f : T \to U$ are non-degenerate nested tuple morphisms, then*

$$\mathsf{coal}(L_{f \oslash g}) = \mathsf{coal}(L_f \oslash L_g).$$

Proof. By Proposition 3.2.6.20, we have

$$\mathsf{coal}(\mathsf{comp}(L_g, \mathsf{size}(L_f))) = \mathsf{coal}(L_{g^c})$$

and we compute

$$\begin{aligned}
\mathsf{coal}(L_f \oslash L_g) &= \mathsf{coal}(L_f \circ (L_g, \mathsf{comp}(L_g, \mathsf{size}(L_f)))) \\
&= \mathsf{coal}(L_f \circ (L_g, L_{g^c})) \\
&= \mathsf{coal}(L_f \circ L_{(g, g^c)}) \\
&= \mathsf{coal}(L_f \circ L_{(g, g^c)}) \\
&= \mathsf{coal}(L_{f \circ (g, g^c)}) \\
&= \mathsf{coal}(L_{f \oslash g}).
\end{aligned}$$

$\square$

Proposition 3.2.6.27. *If f and g are nested tuples and g divides f, then*

$$(f \oslash g)^\flat = f^\flat \oslash^\flat g^\flat.$$

Proof. We compute

$$\begin{aligned}
(f \oslash g)^\flat &= (f \circ (g, g^c))^\flat \\
&= f^\flat \circ (g, g^c)^\flat \\
&= f^\flat \circ (g^\flat \star (g^c)^\flat) \\
&= f^\flat \circ (g^\flat \star (g^\flat)^c) \\
&= f^\flat \oslash^\flat g^\flat.
\end{aligned}$$

$\square$

3.2.6.6 Logical products

In this section, we define the logical product of nested tuple morphisms.

Definition 3.2.6.28. Suppose f and g are nested tuple morphisms. We say f and g are *product admissible* if $\mathsf{codomain}(g) = \mathsf{domain}(f^c)$. If f and g are product admissible we define the *logical product* of f and g to be the nested tuple morphism

$$f \otimes g = (f, f^c \circ g).$$

144

Example 3.2.6.29. The nested tuple morphisms

$$(8,8) \xrightarrow[(1,2)]{f} (8,8,16,16)$$

and

$$(16,16) \xrightarrow[(1,2)]{g} (16,16)$$

are product admissible, and their logical product is

$$((8,8),(16,16)) \xrightarrow[(1,2,3,4)]{f \otimes g} (8,8,16,16).$$

Example 3.2.6.30. The nested tuple morphisms

$$(128,128) \xrightarrow[(3,4)]{f} (32,32,128,128)$$

and

$$(32) \xrightarrow[(2)]{g} (32,32)$$

are product admissible, and their logical product is

$$((128,128),(32)) \xrightarrow[(3,4,2)]{f \otimes g} (32,32,128,128).$$

Proposition 3.2.6.31. *Suppose f and g are non-degenerate nested tuple morphisms and that f and g are product-admissible. Then*

$$L_{f \otimes g} = L_f \otimes L_g.$$

Proof. Suppose $f : S \to T$ and $g : U \to V$ are product admissible, and set

$$L_f^* = \mathsf{comp}(L_f, \mathsf{size}(L_f) \cdot \mathsf{cosize}(L_g))$$

Since f is injective and $\mathsf{codomain}(g) = \mathsf{domain}(f^c)$, it follows that

$$\mathsf{size}(L_f) \cdot \mathsf{cosize}(L_g) \leq \mathsf{size}(S) \cdot \mathsf{size}(V) = \mathsf{size}(T).$$

Using this fact, and the fact that

$$\Phi_{\mathsf{comp}(L_f, \mathsf{size}(T))} = \Phi_{L_{f^c}},$$

we have

$$L_f^* \circ L_g = \mathsf{comp}(L_f, \mathsf{size}(T)) \circ L_g$$
$$= L_{f^c} \circ L_g.$$

Using this fact, we compute

$$L_f \otimes L_g = (L_f, L_f^* \circ L_g)$$
$$= (L_f, L_{f^c} \circ L_g)$$
$$= (L_f, L_{f^c \circ g})$$
$$= L_{(f, f^c \circ g)}$$
$$= L_{f \otimes g}$$

$\square$

Chapter 4

Computations

The categories **Tuple** and **Nest** offer a powerful framework for computing with tractable layouts. It is frequently the case that in practice, however, one comes across tractable layouts A and B that are composable in the context of `cute` but whose standard representations are neither composable in **Tuple** nor **Nest**. This chapter is dedicated to the explication of how one may nevertheless use the categories **Tuple** and **Nest** to compute the *composition*, *logical division*, and *logical product* of tractable layouts, using the notion of **mutual refinement**. We introduce this notion in Section 4.1.1, present an algorithm for computing mutual refinements in Algorithm 4.1.1, and work through many explicit examples.

4.1 Composition

Suppose we want to compute the composition $B \circ A$ of the tractable layouts

$$A = (6,6) : (6,1),$$
$$B = (12,3,6) : (1,72,12).$$

We might try to compute $B \circ A$ by computing the composite of the standard representations f and g of A and B:
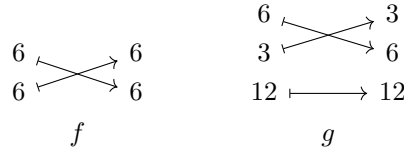
However, these morphisms are not composable, since the codomain $(6,6)$ of f is not equal to the domain $(12,3,6)$ of g. This means that we can not use the morphisms f and g to compute the composite $B \circ A$ directly. We can, however, proceed with our computation by finding a *mutual refinement* of $(6,6)$ and $(12,3,6)$, as depicted below
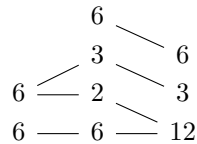
This is a device which converts f and g into composable morphisms f' and g':
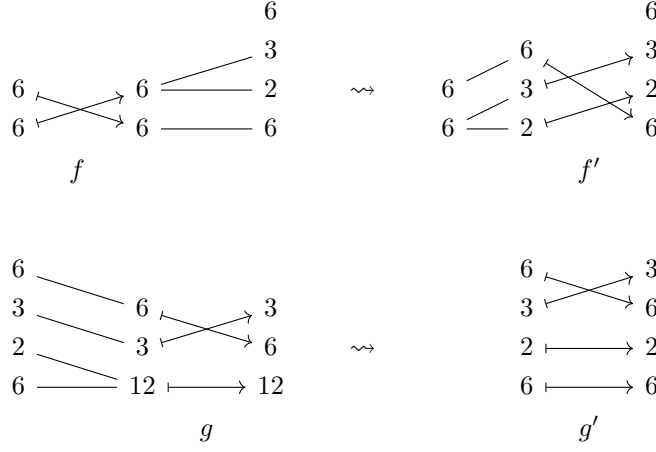
The morphisms f' and g' are composable, so we may form the composite
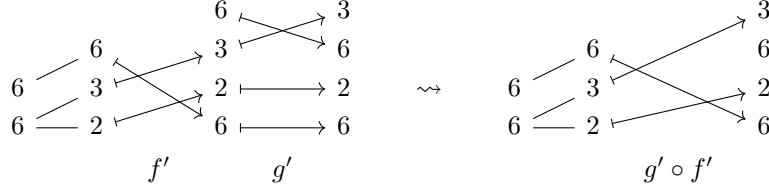
and computing the encoded layout yields

$$B \circ A = L_{g' \circ f'} = ((2,3),6) : ((6,72),1).$$

The goal of this section is to formalize this computational process into an *algorithm* for computing the composite of tractable layouts A and B. As we saw in our example, the non-trivial steps in our computation were

1. finding a mutual refinement of certain (nested) tuples, and

2. using the mutual refinement to convert f and g into composable morphisms f' and g'.

We dedicate the following two sections to the explication of these steps.

4.1.1 Mutual refinements

Before giving a precise definition of mutual refinements using the categorical framework of Chapter 3, we give an informal overview. Consider the tuples $(6,6)$ and $(12,3,6)$ of our motivating example. We asserted that the diagram
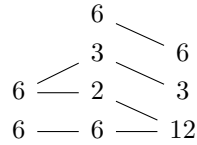
is a mutual refinement of $(6,6)$ and $(12,3,6)$. We can give a more precise description of this mutual refinement as follows. The left half of the diagram represents the refinement $(6,6) \twoheadleftarrow (6,(2,3))$, and

the right half of the diagram represents the refinement $((6,2),3,6) \twoheadrightarrow (12,3,6)$:

$$
\begin{array}{c}
6 \diagup\!\!\!\begin{array}{l} 3 \\ 2 \end{array} \\
6 \!-\! 6
\end{array}
\qquad \rightsquigarrow \qquad
(6,6) \longleftarrow\!\!\!\longleftarrow (6,(2,3))
$$

$$
\begin{array}{c}
6 \\
3 \diagdown\, 6 \\
2 \diagdown\, 3 \\
6 \!-\! 12
\end{array}
\qquad \rightsquigarrow \qquad
((6,2),3,6) \longrightarrow\!\!\!\rightarrow (12,3,6)
$$
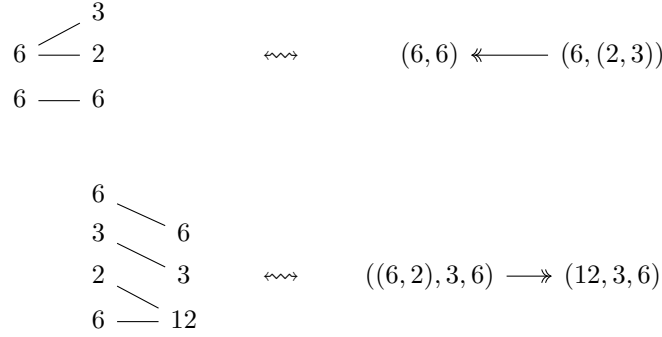
The fact that the two halves of the diagram may be glued together corresponds to the fact that the nested tuple $(6,(2,3))$ *divides* $((6,2),3,6)$, which we denote

$$
(6,(2,3)) \rightarrowtail ((6,2),3,6).
$$

Putting these observations together, we may express our mutual refinement precisely as

$$
\begin{array}{c}
6 \diagdown \\
3 \quad 6 \\
6 \!-\! 2 \diagdown\, 3 \\
6 \!-\! 6 \!-\! 12
\end{array}
\quad \rightsquigarrow \quad
\begin{array}{ccc}
(6,(2,3)) & \rightarrowtail & ((6,2),3,6) \\
\downarrow & & \downarrow \\
(6,6) & & (12,3,6)
\end{array}
$$
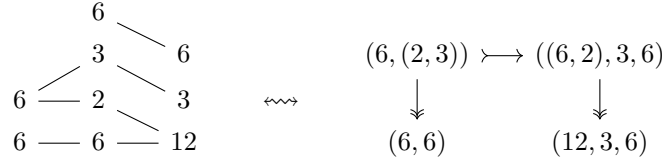
where we opt to depict the refinements $(6,6) \twoheadleftarrow (6,(2,3))$ and $((6,2),3,6) \twoheadrightarrow (12,3,6)$ vertically. We can now give a precise definition of mutual refinements.

Definition 4.1.1.1. Suppose T and U are nested tuples. A *mutual refinement* of (T,U) is a diagram of the form

$$
\begin{array}{ccc}
T' & \rightarrowtail & U' \\
\downarrow & & \downarrow \\
T & & U
\end{array}
$$

Explicitly, this is a pair of nested tuples (T', U') such that

1. T' refines T,

2. U' refines U, and

3. T' divides U'.

Example 4.1.1.2. A mutual refinement of $T = (6,6)$ and $U = (2,6,3)$ is given by

$$
\begin{array}{ccc}
((2,3),(2,3)) & \rightarrowtail & (2,(3,2),3) \\
\downarrow & & \downarrow \\
(6,6) & & (2,6,3)
\end{array}
$$

We depict this mutual refinement as follows.

$$
\begin{array}{c}
6 \diagup\!\!\!\begin{array}{l} 3 \\ 2 \end{array}\!\diagdown\, 3 \\
\qquad 3 \!-\! 6 \\
6 \diagup 2 \!-\! 2
\end{array}
$$

148
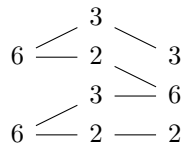
Example 4.1.1.3. A mutual refinement of $T = (8, 8, 8)$ and $U = (2, 8, 8, 8)$ is given by

$$((2,4),(2,4),(2,4)) \longmapsto (2,(4,2),(4,2),(4,2))$$

$$\downarrow \qquad\qquad\qquad\qquad \downarrow$$

$$(8,8,8) \qquad\qquad\qquad (2,8,8,8)$$

We depict this mutual refinement as follows.

Example 4.1.1.4. A mutual refinement of $T = (4, 2, 2, 32)$ and $U = (32, 32)$ is given by

$$(4,2,2,(2,16)) \longmapsto ((4,2,2,2),(16,2))$$

$$\downarrow \qquad\qquad\qquad\qquad \downarrow$$

$$(4,2,2,32) \qquad\qquad\qquad (32,32)$$

We depict this mutual refinement as follows.

Example 4.1.1.5. If $T = (8, 8)$ and $U = (3, 8, 8)$, then there does not exist a mutual refinement of T and U.

Observation 4.1.1.6. In each of the previous examples, we have considered mutual refinements of *flat* tuples T and U. The definition of mutual refinement, however, allows T and U to be any *nested* tuples. In any case, restricting to the flat case is no loss of generality, because there is a one-to-one correspondence between mutual refinements of a pair of nested tuples (T, U), and mutual refinements of their flattenings $(T^\flat, U^\flat)$ (see Lemma 3.2.5.20). In particular, there exists a mutual refinement of (T, U) if and only if there exists a mutual refinement of $(T^\flat, U^\flat)$.

Having made the appropriate definitions, we provide an algorithm for computing a mutual refinement of (T, U).

Algorithm 4.1.1: Mutual refinement algorithm

1 **Input:** Nested tuples T and U.

Algorithm 4.1.1 (continued): Mutual refinement algorithm

2   **Output:** A mutual refinement (T', U') of (T, U), if one exists, else **None**.

---

3   $X \leftarrow T; \ Y \leftarrow U$

4   $X', Y', X_{\mathsf{mode}}, Y_{\mathsf{mode}} \leftarrow ()$

5   $i \leftarrow 1; \ j \leftarrow 1$

6   **while** $i \leq \mathsf{len}(X)$ **and** $j \leq \mathsf{len}(Y)$ **do**

7   **if** $\mathsf{entry}_i(X) = \mathsf{entry}_j(Y)$ **then**

8   append $\mathsf{entry}_i(X)$ to X_{mode}; append X_{mode} to X'; $X_{\mathsf{mode}} \leftarrow ()$

9   append $\mathsf{entry}_j(Y)$ to Y_{mode};

10   append Y_{mode} to Y';

11      $Y_{\mathsf{mode}} \leftarrow ()$

12      $i \leftarrow i + 1$;

13      $j \leftarrow j + 1$

14   **else if** $\mathsf{entry}_i(X)$ divides $\mathsf{entry}_j(Y)$ **then**

15   append $\mathsf{entry}_i(X)$ to X_{mode};

16   append X_{mode} to X';

17      $X_{\mathsf{mode}} \leftarrow ()$

18   append $\mathsf{entry}_i(X)$ to Y_{mode}

19      $\mathsf{entry}_j(Y) \leftarrow \mathsf{entry}_j(Y)/\mathsf{entry}_i(X)$;

20      $i \leftarrow i + 1$

21   **else if** $\mathsf{entry}_j(Y)$ divides $\mathsf{entry}_i(X)$ **then**

22   append $\mathsf{entry}_j(Y)$ to X_{mode};

23   append $\mathsf{entry}_j(Y)$ to Y_{mode};

24   append Y_{mode} to Y';

25      $Y_{\mathsf{mode}} \leftarrow ()$

26      $\mathsf{entry}_i(X) \leftarrow \mathsf{entry}_i(X)/\mathsf{entry}_j(Y)$;

27      $j \leftarrow j + 1$

28   **else**

29   **return None**

30   **end if**

31   **end while**

32   **if** $Y_{\mathsf{mode}} \neq ()$ **then**

33   append $\mathsf{entry}_j(Y)$ to Y_{mode};

34   append Y_{mode} to Y';

35     $j \leftarrow j + 1$

36   **end if**

37   **while** $j < \mathsf{len}(Y)$ **do**

38   append $\mathsf{entry}_j(Y)$ to Y';

39     $j \leftarrow j + 1$

40   **end while**

41   $T' \leftarrow (X')_{\mathsf{prof}(T)}$;

42   $U' \leftarrow (Y')_{\mathsf{prof}(U)}$

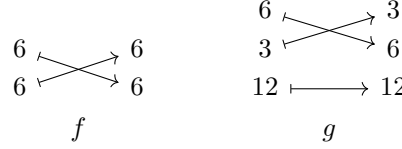43   **return** (T', U')

4.1.2 From mutual refinements to composable morphisms

Recall that in order to compute the composition $B \circ A$ of
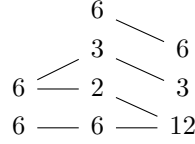
$$A = (6, 6) : (6, 1) \text{ and}$$
$$B = (12, 3, 6) : (1, 72, 12),$$
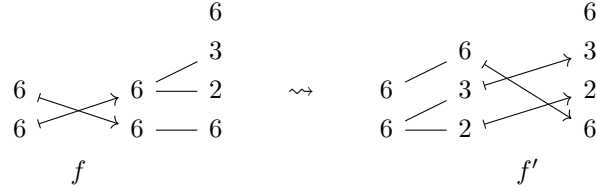
we constructed tuple morphisms
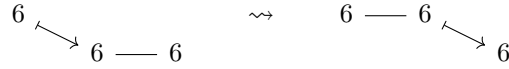

and a mutual refinement.

The next step in our computation is to use our mutual refinement to convert f and g into composable morphisms f' and g'. Before giving a formal, categorical definition of this process, let's illustrate the process with an example.

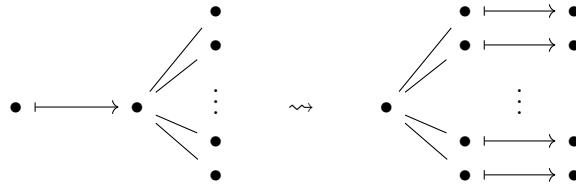We construct f' from f and the left half of our mutual refinement:
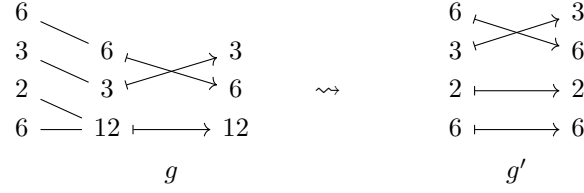

This construction is made by making the replacement

and making the replacement

More generally, we make the replacement

151

The process for constructing g' from g, and the right half of our mutual refinement is similar.
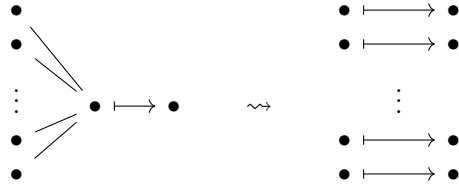
$$
\begin{array}{ccc}
\begin{array}{l}
6 \\
3 \quad 6 \longmapsto 3 \\
2 \quad 3 \qquad 6 \\
6 \longmapsto 12 \longmapsto 12
\end{array}
&
\rightsquigarrow
&
\begin{array}{l}
6 \quad\;\; 3 \\
3 \quad\;\; 6 \\
2 \longmapsto 2 \\
6 \longmapsto 6
\end{array}
\\
g & & g'
\end{array}
$$

This construction is made by making the replacements

$$
6 \relbar\joinrel\relbar 6 \longmapsto 6 \qquad \rightsquigarrow \qquad 6 \longmapsto 6
$$

$$
3 \relbar\joinrel\relbar 3 \longmapsto 3 \qquad \rightsquigarrow \qquad 3 \longmapsto 3
$$

$$
\begin{array}{l}
2 \\
6 \relbar\joinrel\relbar 12 \longmapsto 12
\end{array}
\qquad \rightsquigarrow \qquad
\begin{array}{l}
2 \longmapsto 2 \\
6 \longmapsto 6
\end{array}
$$

More generally, we make the replacement

Having given an informal description of our procedure, we make things precise as follows.

Construction 4.1.2.1. Suppose $f : S \to T$ and $g : U \to V$ are nested tuple morphisms, and (T', U') is a mutual refinement of (T, U). Then we may use the *pullback* and *pushforward* constructions of section 3.2.5 to form the diagram:

$$
\begin{array}{ccccccc}
S' & \xrightarrow{\tilde{f}} & T' & \overset{i}{\rightarrowtail} & U' & \xrightarrow{\tilde{g}} & V' \\
\downarrow & \lrcorner & \downarrow & & \downarrow & \ulcorner & \downarrow \\
S & \xrightarrow{f} & T & & U & \xrightarrow{g} & V
\end{array}
$$

If we set $f' = i \circ \tilde{f}$ and $g' = \tilde{g}$, then

$$
S' \xrightarrow{\;\; f' \;\;} U' \xrightarrow{\;\; g' \;\;} V'
$$

are composable nested tuple morphisms.

4.1.3   The composition algorithm

Algorithm 4.1.2: Tractable layout Composition Algorithm

1   **Input:** Tractable layouts A and B.

Algorithm 4.1.2 (continued): Tractable layout Composition Algorithm

2 **Output:** A weak composite C of A and B, if one exists, else **None**..

---

3 Take the standard representations

$$S \xrightarrow{\quad f \quad} T \qquad\qquad U \xrightarrow{\quad g \quad} V$$

of A and $\mathsf{coal}(B)$, respectively.

4 Use Algorithm 4.1.1 to produce a mutual refinement

$$
\begin{array}{ccc}
T' & \rightarrowtail & U' \\
\downarrow & & \downarrow \\
T & & U
\end{array}
$$

of (T, U). If there does not exist a mutual refinement of (T, U), return **None**.

5 Use Construction 4.1.2.1 to obtain the composable nested tuple morphisms

$$S' \xrightarrow{\ f' \ } U' \xrightarrow{\ g' \ } V'$$

6 Compose f' and g', and compute the encoded layout

$$C = L_{g' \circ f'}$$

7 **return** C

Theorem 4.1.3.1. *If A and B are tractable layouts, then the output C of the previous algorithm is a weak composite of A and B. Consequently,*

$$B \circ A = \mathsf{coal}(C, \mathsf{shape}(A)).$$

Proof. Proposition 3.2.5.15 and tells us that

$$\Phi_{L_{g'}} = \Phi_{L_g} = \Phi_{\mathsf{coal}(B)} = \Phi_B,$$

and Proposition 3.2.5.11 and Example 3.1.3.6 tell us that

$$\Phi_{L_{f'}} = \Phi_{L_f} = \Phi_A.$$

Theorem 3.2.6.21 then implies that

$$
\begin{aligned}
\Phi_C = \Phi_{L_{g' \circ f'}} = \Phi_{g'} \circ \Phi_{f'}^{\mathsf{size}(U')} \\
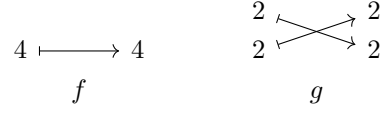= \Phi_B \circ \Phi_A^{\mathsf{size}(B)}.
\end{aligned}
$$

By construction, the shape S' of $L_{f'}$ refines the shape S of A, so we conclude that C is a weak composite of A and B. $\qquad\square$
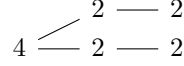
4.1.4 Examples

In this section we illustrate how Algorithm 4.1.3 may be used to compute the composition $B \circ A$ of tractable layouts A and B.

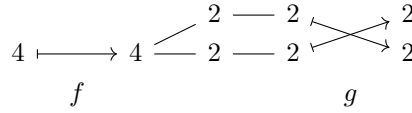**Example 4.1.4.1.** Suppose $A = (4) : (1)$, and $B = (2,2) : (2,1)$.

1. Take the standard representations of A and $\mathsf{coal}(B) = B$.
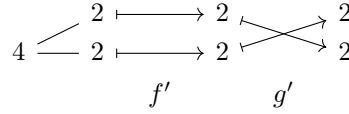
$$
\begin{array}{ccc}
& & 2 \quad\begin{array}{c}\rightarrowtail\\ \times\\ \end{array}\quad 2 \\
4 \longmapsto 4 & & 2 \quad\longmapsto\quad 2 \\
f & & g
\end{array}
$$

2. Apply Algorithm 4.1.1 to obtain the mutual refinement

$$
\begin{array}{c}
\diagup 2 \relbar\relbar 2 \\
4 \relbar 2 \relbar\relbar 2
\end{array}
$$

3. Form the diagram

$$
\begin{array}{c}
\diagup 2 \relbar\relbar 2 \,\begin{array}{c}\rightarrowtail\\ \times\end{array}\, 2 \\
4 \longmapsto 4 \relbar 2 \relbar\relbar 2 \quad 2 \\
f \qquad\qquad g
\end{array}
$$

4. Resolve the diagram

$$
\begin{array}{c}
\diagup 2 \longmapsto 2 \,\begin{array}{c}\rightarrowtail\\ \times\end{array}\, 2 \\
4 \relbar 2 \longmapsto 2 \quad 2 \\
f' \qquad g'
\end{array}
$$

5. Compose f' and g' to obtain

$$
\begin{array}{c}
\diagup 2 \,\begin{array}{c}\rightarrowtail\\ \times\end{array}\, 2 \\
4 \relbar 2 \quad 2 \\
g' \circ f'
\end{array}
$$
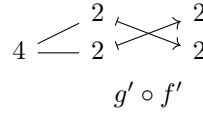
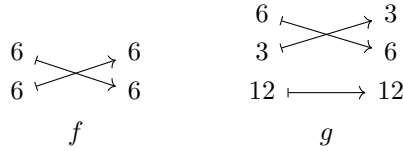6. Compute the associated layout

$$
L_{g' \circ f'} = ((2,2)) : ((2,1)).
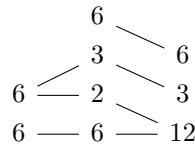$$

7. $L_{g' \circ f'}$ is coalesced over (4), so

$$
B \circ A = ((2,2)) : ((2,1)).
$$

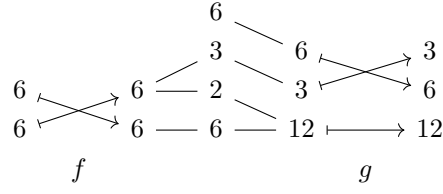**Example 4.1.4.2.** Suppose $A = (6,6) : (6,1)$, and $B = (12,3,6) : (1,72,12)$.

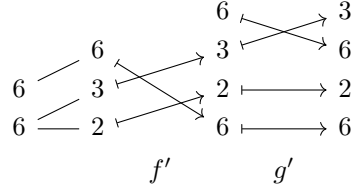1. Take the standard representations of A and $\mathsf{coal}(B) = B$.

$$
\begin{array}{ccc}
& & 6 \,\begin{array}{c}\rightarrowtail\\ \times\end{array}\, 3 \\
6 \,\begin{array}{c}\rightarrowtail\\ \times\end{array}\, 6 & & 3 \quad 6 \\
6 \quad 6 & & 12 \longmapsto 12 \\
f & & g
\end{array}
$$

2. Apply Algorithm 4.1.1 to obtain the mutual refinement

$$
\begin{array}{c}
6 \\
\diagup 3 \diagdown 6 \\
6 \relbar 2 \diagdown 3 \\
6 \relbar 6 \relbar 12
\end{array}
$$

154

3. Form the diagram

$$
\begin{array}{ccccccc}
 & & & 6 & & & \\
 & & & 3 & & 6 & 3 \\
6 & & 6 & 2 & & 3 & 6 \\
6 & & 6 & 6 & & 12 & 12 \\
 & f & & & & g &
\end{array}
$$

4. Resolve the diagram to obtain

$$
\begin{array}{cccc}
 & & 6 & 3 \\
 & 6 & 3 & 6 \\
6 & 3 & 2 & 2 \\
6 & 2 & 6 & 6 \\
 & f' & & g'
\end{array}
$$

5. Compose f' and g' to obtain

$$
\begin{array}{ccc}
 & & 3 \\
 & 6 & 6 \\
6 & 3 & 2 \\
6 & 2 & 6 \\
 & g' \circ f' &
\end{array}
$$
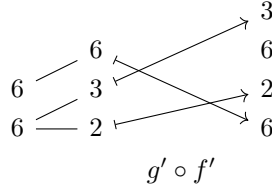
6. Compute the associated layout
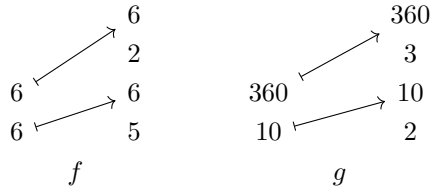
$$L_{g' \circ f'} = ((2,3),6) : ((6,72),1).$$
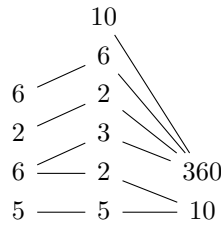
7. $L_{g' \circ f'}$ is coalesced over $(6,6)$, hence

$$B \circ A = ((2,3),6) : ((6,72),1).$$

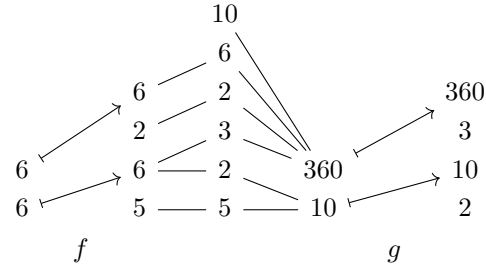**Example 4.1.4.3.** Suppose $A = (6,6) : (5,60)$, and $B = (10,360) : (2,60)$.

1. Take the standard representations of A and $\mathsf{coal}(B) = B$.

$$
\begin{array}{cccc}
 & 6 & & 360 \\
 & 2 & & 3 \\
6 & 6 & 360 & 10 \\
6 & 5 & 10 & 2 \\
 & f & & g
\end{array}
$$

2. Apply algorithm 4.1.1 to obtain the mutual refinement

$$
\begin{array}{ccc}
 & 10 & \\
 & 6 & \\
6 & 2 & \\
2 & 3 & \\
6 & 2 & 360 \\
5 & 5 & 10
\end{array}
$$

155

3. Form the diagram

$$
\begin{array}{c}
10 \\
6 \\
6 \qquad 2 \qquad\qquad 360 \\
2 \qquad 3 \qquad\qquad 3 \\
6 \qquad\qquad 6 \;\text{—}\; 2 \qquad 360 \qquad 10 \\
6 \qquad\qquad 5 \;\text{—}\; 5 \;\text{—}\; 10 \qquad 2 \\
f \qquad\qquad\qquad\qquad g
\end{array}
$$

4. Resolve the diagram to obtain

$$
\begin{array}{c}
10 \\
6 \\
10 \qquad 2 \\
6 \qquad 3 \\
2 \qquad 3 \\
6 \qquad 3 \qquad 2 \\
6 \qquad 3 \qquad 2 \qquad 5 \\
6 \;\text{—}\; 2 \qquad 5 \qquad 2 \\
f' \qquad\qquad g'
\end{array}
$$

5. Compose f' and g' to obtain

$$
\begin{array}{c}
10 \\
6 \\
2 \\
3 \\
3 \\
6 \qquad 2 \\
6 \qquad 3 \qquad 5 \\
6 \;\text{—}\; 2 \qquad 2 \\
g' \circ f'
\end{array}
$$
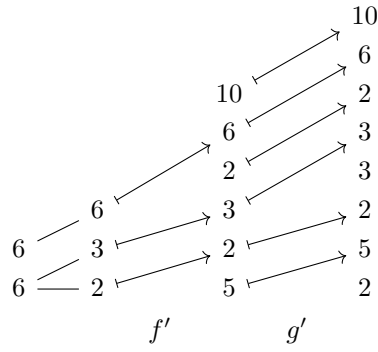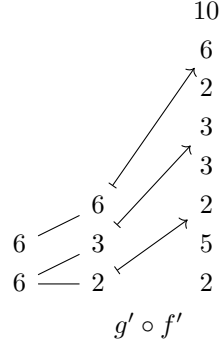
6. Compute the associated layout

$$
L_{g' \circ f'} = ((2,3),6) : ((10,60),360).
$$

7. The layout $L_{g' \circ f'}$ is coalesced over $(6,6)$, so

$$
B \circ A = ((2,3),6) : ((10,60),360).
$$

Appendix A

An introduction to categories

Throughout this work, we freely use the language of *categories* which are mathematical objects which abstract the notion of *morphisms* and their *composition*. The purpose of this appendix is to provide a concise and user-friendly introduction to the basics of categories. In particular, we aim to the answer the following questions:

1. What is a category?

2. What is a functor?

Those capable of answering these questions with confidence, and with examples in mind, will be able to understand the most important conepts and constructions in the current work. For those interested in learning the more advanced concepts from category theory, such as *natural transformations*, *adjunctions*, and *(co)limits*, we recommend [1].

A.1 What is a category?

We begin by addressing the first question. Before giving a definition, let's consider a motivating example. Suppose X and Y are sets. A *function* $f : X \to Y$ assigns to each element $x \in X$ some element $f(x) \in Y$. We refer to X as the *domain* of f and to g as the *codomain* of f.

Example A.1.0.1. There is a function $f : \mathbb{Z} \to \mathbb{Z}$ given by

$$f(x) = 2x.$$

Example A.1.0.2. There is a function $g : \mathbb{Z} \to \mathsf{Bool}$, where $\mathsf{Bool} = \{\mathbf{True}, \mathbf{False}\}$, given by

$$g(x) = \begin{cases} \mathbf{True} & x \text{ is even,} \\ \mathbf{False} & x \text{ is odd.} \end{cases}$$

If $f : X \to Y$ and $g : Y \to Z$ are functions, then we can *compose* f and g: The composite of f and g is the function $g \circ f : X \to Z$ given by

$$(g \circ f)(x) = g(f(x)).$$

Example A.1.0.3. If f and g are the functions of Examples A.1.0.1 and A.1.0.2, then the composite $g \circ f : \mathbb{Z} \to \mathsf{Bool}$ is given by

$$(g \circ f)(x) = \mathbf{True}.$$

Composition of functions satisfies two essential properties. First, composition is *associative*: if f and g are composable, and g and h are composable, then

$$h \circ (g \circ f) = (h \circ g) \circ f.$$

Second, every set X has an *identity* function $\mathsf{id}_X : X \to X$ given by

$$\mathsf{id}_X(x) = x.$$

If $f : X \to Y$ is any function, then precomposing with id_X or post-composing with id_Y leaves the function f unchanged:

$$f \circ \mathsf{id}_X = f = \mathsf{id}_Y \circ f.$$

In pure and applied mathematics, there are many instances where we have some collection of **objects**, and **morphisms** between those objects, which have the same formal behavior of sets and functions: morphisms can be composed in an associative fashion, and objects admit identity morphisms. While functions between sets are the prototypical example, the objects in a category need not be sets, and the morphisms in a category need not be functions. We will see many such examples later on. To capture this recurring structure, we define the notion of a **category**.

Definition A.1.0.4. A *category* **C** consists of

1. a collection of *objects*:
$$\mathsf{ob}(\mathbf{C}) = \{X, Y, Z, \dots\}.$$

   These objects may be sets, tuples, numbers, vector spaces, matrices, or some other mathematical structure, depending on the category **C**.

2. a collection of *morphisms* between those objects:
$$\mathsf{mor}(\mathbf{C}) = \{f, g, h, \dots\}.$$

   Each morphism $f : X \to Y$ in **C** has a *domain X* and a *codomain Y*, which are objects in **C**.

3. a *composition rule*: If $f : X \to Y$ and $g : Y \to Z$ are morphisms in **C**, then there is a morphism
$$g \circ f : X \to Z$$

   called the *composite* of f and g. Composition of morphisms in **C** is *associative*, in that
$$h \circ (g \circ f) = (h \circ g) \circ f,$$

   when defined.

4. *identity morphisms*: If X is an object in **C**, then there is a morphism
$$\mathsf{id}_X : X \to X$$

   called the *identity morphism* on X. If $f : X \to Y$ is any morphism in **C**, then
$$f \circ \mathsf{id}_X = f = \mathsf{id}_Y \circ f.$$

Let's take a look at some important examples of categories. We begin with the motivating example.

Example A.1.0.5. There is a category **Set** whose objects are sets, and whose morphisms are functions. The composition of morphisms is given by functional composition:

$$(g \circ f)(x) = g(f(x))$$

and the identity morphism on a set X is the identity function

$$\mathsf{id}_X(x) = x.$$

Example A.1.0.6. There is a category **Vect** whose objects are the vector spaces $\mathbb{R}^n$ for $n \geq 0$, and whose morphisms are *matrices*. Specifically, a morphism

$$A : \mathbb{R}^n \to \mathbb{R}^m$$

in **Vect** is a $m \times n$ matrix A. Composition in **Vect** is given by taking matrix products:

$$B \circ A = BA,$$

and the identity morphism on $\mathbb{R}^n$ is the $n \times n$ matrix

$$\mathsf{id}_{\mathbb{R}^n} = I_n = \begin{bmatrix} 1 & 0 & \cdots & & 0 \\ 0 & 1 & & & \\ \vdots & & \ddots & & \vdots \\ & & & 1 & 0 \\ 0 & & \cdots & 0 & 1 \end{bmatrix}.$$

Example A.1.0.7. There is a category **Div** whose objects are integers $a \geq 1$, and in which there is a unique morphism

$$\mathsf{div}_a^b : a \to b$$

if a divides b. If a divides b and b divides c, then a divides c, which means that we have a well defined composition rule

$$\mathsf{div}_b^c \circ \mathsf{div}_a^b = \mathsf{div}_a^c,$$

and the identity morphism

$$\mathsf{id}_a = \mathsf{div}_a^a$$

exists since every positive integer a divides itself.

In addition to the definition of a category, there are a few important categorical concepts that we need to understand. For instance, it is important to understand the notion of an *isomorphism*, which generalizes the notion of a *bijection* of sets.

Definition A.1.0.8. Suppose **C** is a category, and suppose $f : X \to Y$ is a morphism in **C**. We say f is an *isomorphism* if there exists a morphism $f^{-1} : Y \to X$ in **C** such that

1. $f^{-1} \circ f = \mathsf{id}_X$, and

2. $f \circ f^{-1} = \mathsf{id}_Y$.

Example A.1.0.9. In the category **Set**, an isomorphism is a *bijection*: a function $f : X \to Y$ such that for each $y \in Y$, there exists a unique $x \in X$ with $f(x) = y$. For example, the function $f : \mathbb{Z} \to \mathbb{Z}$ given by

$$f(x) = x + 10$$

is a bijection, with inverse $f^{-1} : \mathbb{Z} \to \mathbb{Z}$ given by

$$f^{-1}(x) = x - 10.$$

Example A.1.0.10. In the category **Vect**, an isomorphism is an *invertible matrix*. For example, the matrix

$$A = \begin{bmatrix} 3 & 2 \\ 1 & 1 \end{bmatrix}$$

is invertible with inverse

$$A^{-1} = \begin{bmatrix} 1 & -2 \\ -1 & 3 \end{bmatrix}$$

since

$$A^{-1}A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = AA^{-1}$$

Example A.1.0.11. In the category **Div**, the only isomorphisms are the identity morphisms

$$\mathsf{id}_a = \mathsf{div}_a^a.$$

This is because if a divides b and b divides a, then $a = b$.

A.2 What is a functor?

Next, we turn our attention to the second question.

Definition A.2.0.1. Suppose **C** and **D** are categories. A *functor* $F : \mathbf{C} \to \mathbf{D}$ consists of

1. for each object X in **C**, an object FX in **D**, and

2. for each morphism $f : X \to Y$ in **C**, a morphism

$$Ff : FX \to FY$$

   in **D**,

satisfying the following properties:

1. F is compatible with composition: If f and g are composable morphisms in **C**, then

$$F(g \circ f) = Fg \circ Ff.$$

2. F is compatible with identities: If X is an object in **C**, then

$$F\mathsf{id}_X = \mathsf{id}_{FX}.$$

Example A.2.0.2. There is a functor $F : \mathbf{Div} \to \mathbf{Set}$ defined as follows. On objects, F is given by

$$Fa = [0, a] = \{x \in \mathbb{R} \mid 0 \le x \le a\}.$$

and on morphisms, F is given by

$$F\mathsf{div}_a^b(x) = \tfrac{b}{a} \cdot x.$$

Let's verify that F is a functor.

1. F is compatible with composition: If a divides b and b divides c, then

$$(F\mathsf{div}_b^c \circ F\mathsf{div}_a^b)(x) = F\mathsf{div}_b^c(F\mathsf{div}_a^b(x)) = \frac{c}{b} \cdot \left(\frac{b}{a} \cdot x\right) = \frac{c}{a} \cdot x = F\mathsf{div}_a^c(x).$$

2. F is compatible with identities: If $a \ge 1$, then

$$F\mathsf{id}_a(x) = F\mathsf{div}_a^a(x) = \frac{a}{a} \cdot x = \mathsf{id}_{Fa}(x).$$

Bibliography

[1] Saunders Mac Lane. *Categories for the Working Mathematician.* 2nd ed. Vol. 5. Graduate Texts in Mathematics. Springer, 1998. ISBN: 978-0-387-98403-0.